NO. 43 $2.50 DECEMBER 1981

MICRO™

THE 6502/6809 JOURNAL

Applesoft Feature

Pascal Tutorial, Part 2

OSI Symbolic Disassembler

PET "Listener" Atari Joysticks

Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT CONSUMER PRODUCTS

Microsoft Consumer Products, 400 108th Ave. N.E., Bellevue, WA 98004. (206) 454-1315


ADVENTURE has gone GOLD!
AND WE WANT YOU TO CELEBRATE WITH US.*
The 12 Scott Adams' Adventures are presented in our **Limited Gold Edition.** Each tape and disk is individually numbered, and guaranteed until July 10th, 2001. Each package contains a certificate of authenticity, a registration card and an autographed, (rather interesting) photo of the author.
The 12 Adventures normally retail for $239.40 individually on tape and $159.80 for 4 triple-pack disks.
The **Limited Gold Edition** is yours . . . **forever** . . . for $100.00 tape or disk, value for value.
To Order: The **Limited Gold Edition** is available in very limited quantity, from interested Software Retailers. Ask your local dealer. If he does not have The **Limited Gold Edition** . . . Then call toll free 1-800-421-5770. In California 1-800-262-4242 (Local 213-670-9461) . . . And we will direct you to a dealer who does have the **Limited Gold Edition.** *Supply Is Limited!*
Limited Gold Edition
Scott Adams' Adventure Series
Autographed and Individually Numbered
The **Limited Gold Edition** is available in the following different configurations.
• Apple 2 Disk with TRS-80 Disk in same package
• Atari Tape
• Atari Disk
• TRS-80 Tape
$100.00 Each
*In keeping with the spirit of Adventure, $1,000.00 in Gold Coins is hidden within The **Limited Gold Edition.** Its waiting for **you.**
The **Limited Gold Edition** from Adventure International is distributed exclusively by:
SOFTSEL
8295 SO. LA CIENEGA BLVD. • INGLEWOOD, CA 90301

NO. 43 DECEMBER 1981

MICRO™

THE 6502/6809 JOURNAL

DEPARTMENTS

HARDWARE

PROGRAMMING AIDS

PROGRAMMING LANGUAGE

APPLESOFT


"THIEF"
"The damn things nearly killed me."
THIEF
It seemed like a cinch assignment. At least for the galaxy's master thief.
Just sneak inside the Star-Complex Base, dodge a few guardian robots and grab the formula. A piece of cake.
Oh yeah! They didn't say the robots went berserk at the sight of a human. They didn't say I'd have to laser my way through 7 levels. Or, that rampaging robots would keep coming and coming.
And most of all, they didn't bother to say there's some weird blob, an unearthly something that tracks you right thru the damn walls.
I made it to the 7th level. I found the formula. But, more robots are massing out there. And I'm wounded. My energy charge is low.
You. Yes, you. How good a Thief are you? Prove yourself. Come get me. Come save me.
$29.95 for Apple II*. At computer stores, or from DATAMOST.
VISA/MASTERCHARGE accepted. $1.00 shipping/handling charge. (CA residents add 6% tax)
*Apple II is a trademark of Apple Computer, Inc.
DATAMOST
19273 Kenya St., Northridge, CA 91326, (213) 366-7160

About the Cover

At first glance our cover looks "traditional" for this time of year. Actually, we have broken our own tradition of using a color photo and monochrome computer graphic overlay. This month's cover photo depicts winter in the Berkshire Mountains of Massachusetts.

The graphic overlay was created using a PET and the programmable character feature of the CBM 4022 printer. The colors were stripped in by Gilbert Color Labs of Hudson, New Hampshire.

(Cover Photo by John Rodriguez)

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and Avon, MA 02322
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to
MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC

Subscription rates	*Per Year*
U.S.	$18.00
Foreign surface mail	$21.00
Air mail:	
Europe	$36.00
Mexico, Central America	$39.00
Middle East, North Africa	$42.00
South America, Central Africa	$51.00
South Africa, Far East, Australasia	$60.00

MICRO Editorial

New Product Review Policy

As you may be aware, MICRO has not published product reviews for several months. Previously, MICRO handled reviews with MICROScope. This system involved receiving a product, selecting a qualified reviewer, sending the reviewer the product, preparing the review, sending a copy to the manufacturer, and again contacting the reviewer. By the time the review finally appeared in the magazine, six months might have elapsed from our initial receipt of the product. This ended up being unsatisfactory for both the manufacturer and for the readers. Furthermore, many competitive products weren't covered, and we had trouble reviewing all the products received. As a result of the long delays and the inherent bias, we stopped publishing reviews altogether.

We do realize that both manufacturers and readers want reviews, even if they don't meet such high standards of impartiality as we had intended with MICROScope. As a result, we are changing and relaxing our review policy.

Reviews will be considerably shorter than MICROScopes tended to be (probably a half column) and will contain material that is primarily evaluative rather than descriptive in nature. (You can get the details by reading the manufacturer's promotional literature, advertisements, entries in our Hardware and Software Catalogs, or by listening to the salesman.) After you have that information, you need advice from a knowledgeable person on whether to buy the product. Our reviewers will tell you the answers to such vital questions as:

Does it work?
Is it well documented?
What won't it do?
Is it "user-friendly"?
Do I need a degree in computer science to use it?

We are still in the process of assembling our review panel, so it will be a few months before any reviews appear in the magazine. In the meantime, we think it is important to let both the manufacturers and the readers know that we are not ignoring this very important area of service.

Coming Changes

Next month, to start the year off right, we'll introduce some changes in MICRO.

A better quality of printing and more use of color within the magazine will be two obvious enhancements. Instead of the current binding where the pages are glued together (called "Perfect"), we will change back to the form with the staples (called "Saddle-stitch"). This makes it easier for the magazine to lie flat while you are entering a program. Also, the magazine will no longer have the holes for three-ring binders. We expect a few complaints about this, but the magazine has grown so in size that now it is difficult to collect more than four issues in a standard binder. Furthermore, hole punching is an extra step, which has several times significantly delayed our shipping.

MICRO will no longer be sent to subscribers in envelopes. Instead, the mailing label will be pasted directly onto the cover of the magazine. Stuffing envelopes, too, has delayed mailings, and we suspect that the post office may not have recognized the envelopes as magazines, denying them the high priority they deserve. (To increase durability, we are increasing the weight of the cover. At the same time we are decreasing the weight of the paper inside, so that the overall weight will not change significantly.)

We expect these changes will get your MICRO to you sooner in an even more useful form, while helping to keep costs as low as possible.

Loren W. Wright

MICRO New Publications

Mike Rowe
New Publications
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Games

PET Games and Recreations by Mac Oglesby, Len Lindsay, and Dorothy Kunkin. Reston Publishing Company, Inc. (Reston, Virginia), 1981, vii, 245 pages, diagrams, drawings, listings, 7 × 9 inches, paperbound.
ISBN: 0-8359-5530-3
ISBN: 0-8359-5529-X (pbk.) $12.95

This collection of games, designed to entertain and educate, is suitable for both beginning programmers and computer veterans. Each game is accompanied by a brief summary, including instructions, background information, and level of strategy.

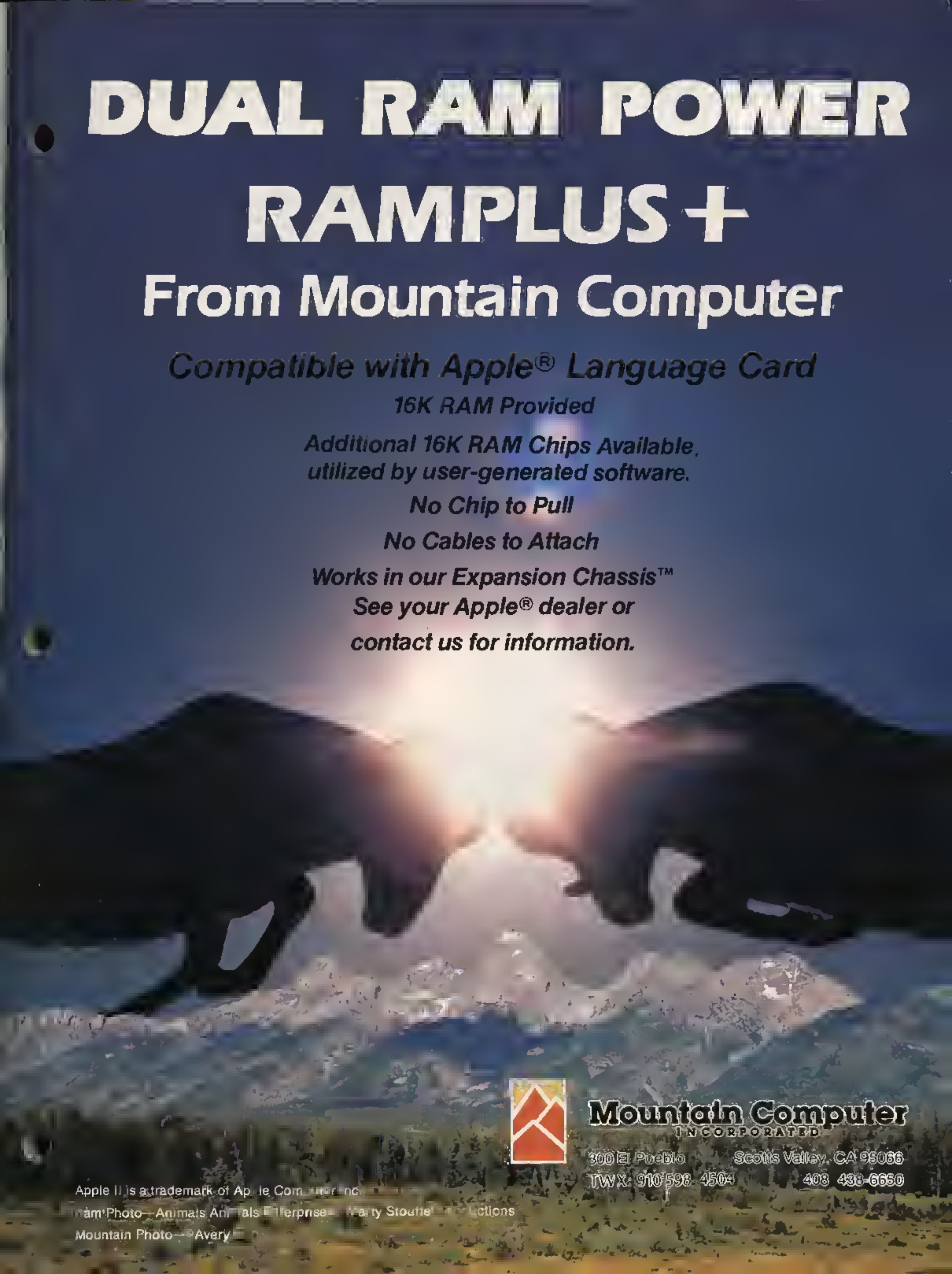
CONTENTS: *How To Use This Book for Fun and Learning*—The Games; The Game Write-ups; The Listings; BASIC for Beginners; Special Guest Lectures; Games Bibliography. *Plan-Ahead Games*—Qwert; Capture; Tic Tac Toe; Reverse; Watchperson; Square; Motie; Sinners; Brainbuster. *Games of Deductive Reasoning*—Stars; Button, Button; Hurkle; Martian Hunt; The Code Game; Dr. Factor. *Games of Chance*—In Between (Accy Deuccy); Thrice Dice. *Language and Counting Skills Games*—How Many?; Crossword Puzzle (Puzzlebox, Puzzle Entry); Wordsearch—A Hunt for Hidden Words. *Recreations*—Bouncing Ball Track Ways; Hypername, Nameblinker, Namerunner; Happy Birthday; Starfill; Marblestat; Petsketch. *Special Guest Lectures. Games Bibliography.*

Inside BASIC Games by Richard Mateosian. Sybex Inc. (2344 Sixth St., Berkeley, California 94710). 1981, xx, 325 pages, illustrations, 7 × 9 inches, paperbound.
ISBN: 0-89588-055-5 $14.95

In this book, the author uses eight different kinds of computer games to teach interactive programming in BASIC. The book is written for people interested in designing original games programs.

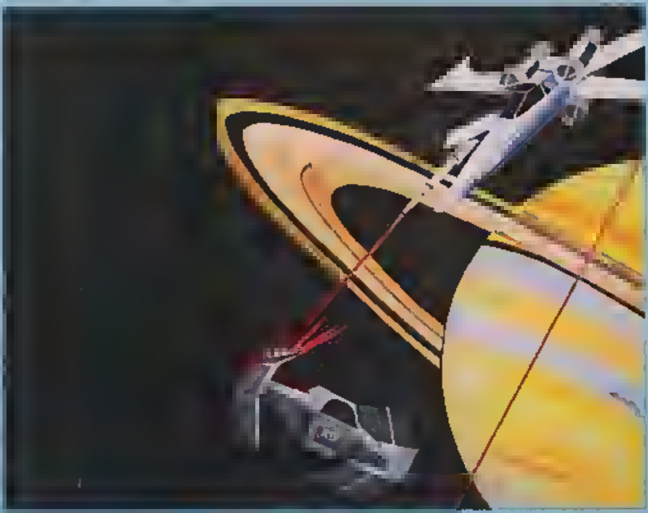
CONTENTS: *Arithmetic Games*—Addition Drill; The Addition Drill Program; Arithmetic Drill; The Arithmetic Drill Program; Possible Additions and Changes; Summary. *Guessing Games*—General Form of Guessing Games; Four; A Sample Game; The Guessing Game Program; The Hangman Program; Possible Additions and Changes; Summary. *Time Games*—The Pet Clock; Clock; The Clock Program; Card Memory; The Card Memory Program; Ten-Key Flicker; Timer; Summary. *Date Games*—Birthday; The Birthday Program; Calendar; The Calendar Program; Summary. *Taxman*—Instructions for Taxman; The Taxman Program; Suggestions for Improvements and Additions; Summary. *Programming with Free BASIC*—Program Design Techniques; Free BASIC; Translating from Free BASIC into BASIC; Free BASIC, Structured Programming and Pascal; Summary. *The Match-Up Game*—The Game-Building Phase; The Playing Phase; The Match-up Program; Changes and Improvements; Summary. *Craps*—Instructions for Craps; The Craps Program; Suggested Additions and Improvements; Summary. *Alien Life*—Alien Encounter; The Rules of Game of Life; The Alien Life Program; Improvements and Additions; Summary. *Appendix A. Index.*

MICRO



DUAL RAM POWER
RAMPLUS+
From Mountain Computer
Compatible with Apple® Language Card
16K RAM Provided
Additional 16K RAM Chips Available,
utilized by user-generated software.
No Chip to Pull
No Cables to Attach
Works in our Expansion Chassis™
See your Apple® dealer or
contact us for information.
Mountain Computer
INCORPORATED
300 El Pueblo
Scotts Valley, CA 95066
TWX: 910 598-4504
408 438-6650
Apple II is a trademark of Ap le Com
Mountain Photo—©Avery

NOW FROM SYNERGISTIC SOFTWARE



ARCADE **NEW** GAME

ESCAPE FROM ARCTURUS

A tast action dual mode high-res arcade style game with excellent color graphics and sound effects. In command ot the Space Fortress you fend off powerful attackers from all sides. The attacking Griplems use several differenl ships, force tields and weapons in ever increasing numbers and speed. Once you take charge of the Arcturon ESCAPE ships you must battle a variety ot invading Griplem landing craft. Control your ship with paddle or joystick while trying to ESCAPE into hyperspace. Provides more variety, challenge and choices than other arcade games.

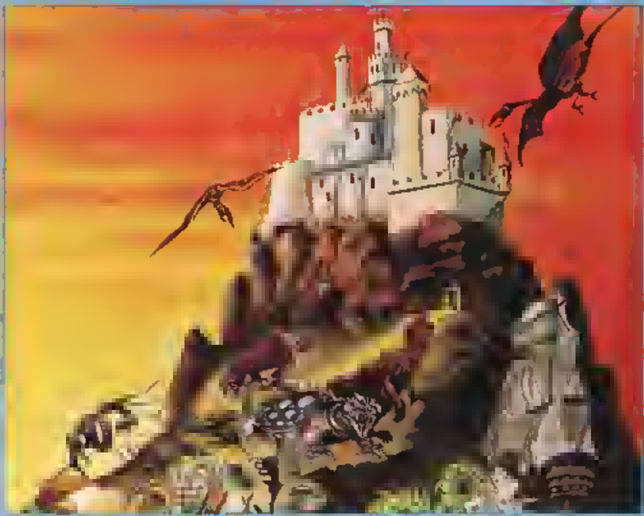
Full Color – High Speed – Machine Language
Requires 48K, disk with DOS 3.3, Applesoft and paddles.
$35.00

ADVENTURE GAME

ODYSSEY:
THE COMPLEAT APVENTURE
NOW AVAILABLE IN APPLESOFT or INTEGER

A mythic advenlure game utilizing the Apple's colorful high-res graphics. The adventure is set in the Sargalo Sea, a haunted realm of sea serpents, dragons, and fabulous beasts. Many islands provide an endless variety of hazards and loes as you explore villages, castles, and dungeons. You walk, ride, sail and tly across detailed jungles, swamps, mountains and seas. At every step you decide the actions to be taken and your party's nature. You will collect magical devices, gather an army and set out on a quest to attack the powerful fortress on Lapour. Always challenging and uniquely different each time you play.

Requires 48K, disk with DOS 3.3, Applesoft or Integer.
$30.00



EDUCATIONAL **NEW** SOFTWARE

THE PLANETARY GUIDE

This colorful high-resolution handbook puts the solar system at your fingertips. The Planetary Guide is a new and exciting way to learn astronomy as you follow the program from lhe moon and sun, to the planets and their orbits, to comets and asteroids. All major solar system members are displayed in detail. Moon phases and planetary movement are animated on high-res screens. Pick your date and see the location of all planets in orbit as well as where each planet is listed in each constellation. Graphic displays along with text data and detailed tables provide as well as where each planet appears in each constellation keystroke commands allow rapid access to any of the general purpose or detailed programs.

Great with The Star Gazer's Guide.
Requires 48K disk with DOS 3.3, Applesoft
$30.00

Synergistic Software
5221 120th Ave. S.E.
Bellevue, WA 98006
206-226-3216

All programs require Apple II or Apple II Plus. See your dealer tor these and other tine Synergistic Software products or order directly.

Data Collection with Your Micro

This article describes how to construct and implement an interface which enables high-speed sampling and recording of experimental data. Written for an AIM 65, it is readily adapted to any 6502 microprocessor with either a 6520 or 6522 interface adapter.

John C. Traeger
Dept. of Physical Chemistry
La Trobe University
Bundoora, Victoria 3083
Australia

One application for which the microprocessor is ideally suited is the rapid measurement of experimental data. This article describes how an AIM 65, with the aid of an inexpensive (less than $50) and relatively simple interface, can be made to function as a high-speed recorder. The interface and operating program can be readily adapted for use on other 6502 systems.

The basic interface consists of a 10-bit analog-to-digital converter (ADC) which is connected to the AIM 65 *via* the user-dedicated 6522 Versatile Interface Adapter. To simplify design, an Analog Devices AD571 converter is used to digitize the signal voltage to a relative accuracy of 0.1%. This device is a successive approximation ADC consisting of a 10-bit digital-analog converter, voltage reference, clock, comparator, successive approximation register and output buffers all contained on a single chip. For less critical applications, a cheaper version with only 0.4% guaranteed accuracy (AD570) could be employed. Both converters have a typical conversion time of 25 μ sec.

Digital Interface

The complete circuit for interfacing to the AIM 65 Application connector (J1) is shown in figure 1. Data transfer between the ADC and the microprocessor is *via* the two data ports of the 6522: PA0-PA7 and PB1-PB2 are used for the 10 data bits, with PB7 being used to monitor the DATA READY (end of conversion) status line of the AD571. The remaining unused bits of port B are held at logic 0 to reduce software overhead time.

Initiation of a conversion is triggered by a positive pulse on the BLANK and CONVERT line of the AD571. Because the pulse width must be greater than 2 μ sec, this is best accomplished under program control using the CA2 control line of the 6522 in the manual mode to generate a 6 μ sec-wide positive pulse. The two CB control lines can be used to synchronize the timing of the data acquisition with the signal to be measured.

Analog Interface

The input voltage levels and polarities to the AD571 are determined by the bipolar offset control pin. A unipolar 0 to +10V range is obtained if this pin is shorted to digital common, and a bipolar ±5V range with offset binary output code results if the pin is left open. Because the AD571 has a relatively low input resistance (5 Kohms), it is necessary to buffer the analog input. Although the present circuit has achieved this by using an LM308 operational amplifier on the input in a voltage follower configuration, it is possible to include a variable gain option. The 15 Ω resistor in series with the analog input to the AD571 gives a typical full scale calibration error of ±0.2%. For a more precise calibration it is necessary to replace this with a 50 Ω trimmer. In addition to the +5V supply used for the AIM 65, a ±15V power supply is required for operation of the LM308 and AD571. (A ±12V supply could be satisfactorily used.) To minimize the effects of noise and interference, each power-supply line is bypassed to ground right at the converter with a 4.7 μ F tantalum capacitor.

The maximum signal frequency that can be handled with less than 1 least significant bit (LSB) error due to timing is given by

$$f_{max} = 2^{-n}/2 \pi T_c$$

where n is the number of bits and Tc is the conversion time. Thus, for the AD571 connected as shown in figure 1, it is only possible to measure signals with frequencies of 6 Hz or less to an accuracy of 0.1%. An increase in the frequency response can only be achieved at the expense of overall accuracy. The reason for this poor frequency response is that the input voltage to a successive approximation ADC must remain constant to within 1 LSB during the conversion process (25 μsec for the AD571).

It is possible to greatly increase the frequency response by incorporating a sample and hold amplifier (SHA) in place of the buffer amplifier. This enables a constant input to the ADC to be maintained during a conversion which represents the analog signal as of a certain precisely known time. The ultimate limitation on timing accuracy is the aperture jitter or uncertainty of the SHA.

When a SHA is used with an ADC, the timing uncertainty of the conversion process is reduced by the ratio of aperture jitter, T_A, to the conversion time (i.e. T_A replaces T_c in equation 1). For example, if an Analog Devices AD528 SHA, which has an aperture uncertainty of 15 nsec, is used in conjunction with the AD571, the maximum frequency

signal that can he digitized to 1LSB error is 10 kHz. However, in order to faithfully reproduce a signal, it is necessary to sample it at a rate which is at least twice the highest frequency. Thus, for a 10 kHz signal, the microcomputer must be capable of acquiring data in less than 50 μ sec. This is about the limit at which 10-bit data can be collected with a 1MHz 6502 microprocessor.

Many natural phenomena have an exponential, rather than an oscillatory, nature. For example, the discharging of a capacitor, various chemical reactions, and the decay of radioactive isotopes all follow an exponential relationship. These processes are characterized by a particular parameter called the half-life, which is the time taken for the initial quantity to be reduced by 50%. The fastest exponential decay that can be followed digitally with less than 1LSB error is given by

$$T_{1/2}\min = 0.6931\ T_C 2^n$$

where n is the number of bits of precision and Tc is the conversion time of the ADC (or the aperture uncertainty if a SHA is used on the input). The AD571 as shown in figure 1 is capable of accurately following processes with a half-life of only 18 msec.

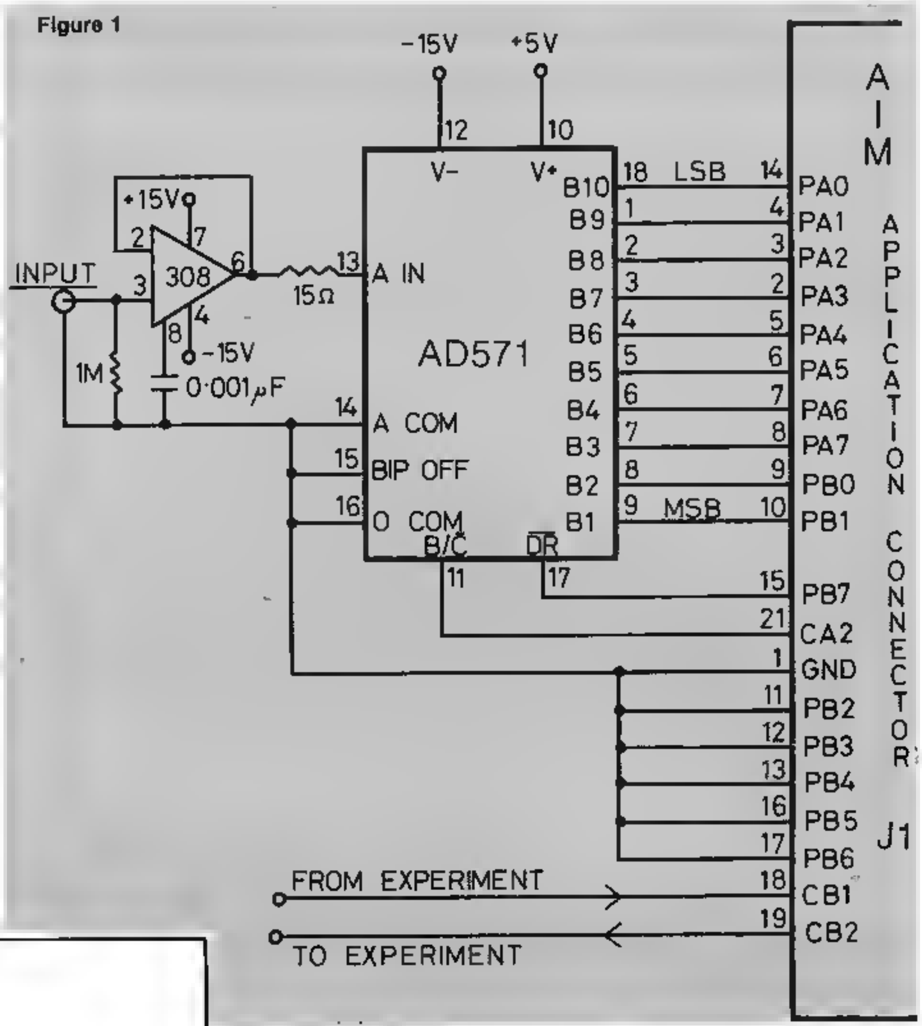


Figure 1

```
;*******************
;*                 *
;*     DATACQ      *
;*                 *
;* BY J.C. TRAEGER *
;*                 *
;*******************
;
;COLLECTS DATA FROM AN AD571 ANALOG-TO-
;DIGITAL CONVERTER INTERFACED TO AN AIM 65
;VIA THE USER 6522 VIA.  ALL DATA IS
;STORED IN MEMORY FOR SUBSEQUENT PROCESSING
;BY THE BASIC PROGRAM.  THE TIME INTERVAL
;BETWEEN SUCCESSIVE POINTS IN UNITS OF
;0.01 SEC. IS PASSED TO THE PROGRAM VIA
;THE USR ARGUMENT AS AN INTEGER BETWEEN 1 AND 255
;
CTR     EPZ $A9              ;COUNTER FOR NO. OF 0.01 SEC INTERV.
BUFFLO  EPZ $AA              ;LOW ORDER BYTE OF ADC BUFFER
BUFFHI  EPZ $AB              ;HIGH ORDER BYTE OF ADC BUFFER
MSEC10  EPZ $AD              ;LOW ORDER BYTE OF USR ARGUMENT
NCONV   EPZ $AE              ;CONVERSION COUNT BUFFER
;
DATLO   EQU $0F80            ;START ADDR. FOR LOW BYTE OF DATA
DATHI   EQU DAOA             ;START ADDR. FOR HIGH BYTE OF DATA
;
;6522 REGISTER DEFINITIONS
;
IRB     EQU $A000            ;INPUT REGISTER B
IRA     EQU $A001            ;INPUT REGISTER A
T1CL    EQU $A004            ;TIMER1 LOW ORDER COUNTER
T1CH    EQU $A005            ;TIMER1 HIGH ORDER COUNTER
ACR     EQU $A00B            ;AUXILLIARY CONTROL REGISTER
PCR     EQU $A00C            ;PERIPHERAL CONTROL REGISTER
IFR     EQU $A00D            ;INTERRUPT FLAG REGISTER
;
IFIX    EQU $BEFE            ;INTEGER CONV.--BASIS SUBR.
;
        ORG $F00
        OBJ $800
```

Software

The sample program shown here demonstrates the ability of the AIM 65 to follow fast reactions. For convenience, a minimum time interval of 10 msec. between data points has been chosen with a total of 50 points. It is a simple process to modify the program to accommodate more or less data points. The program enables reactions with a half-life between 0.1 sec. and 25 sec. to be satisfactorily followed.

Because of the speed with which data is collected, and for accurate control of the interval timing, the data acquisition section has been written as an assembly code subprogram (DATACQ). This places the data directly into memory for subsequent processing by the main BASIC program. The subprogram occupies less than 256 bytes, including data storage. The time interval, in units of 0.01 sec., is passed to DATACQ as an 8-bit integer (MSEC10). It is necessary to ensure that the USR argument is between 1 and 255 as no check is made within DATACQ. The page zero locations used by DATACQ

```
                   ;
0F00 20FEBE        DATACQ JSR IFIX          ;GET THE USR ARGUMENT INTO $AD
0F03 A940                 LDA #$40          ;SET UP FOR A MAXIMUM OF
0F05 85AE                 STA NCONV         ; 64 CONVERSIONS/POINT
0F07 A5AD                 LDA MSEC10
0F09 C5AE                 CMP NCONV         ;MORE THAN 0.63 SEC./POINT?
0F0B B004                 BCS START         ;YES, LEAVE NCONV=64
0F0D 6901                 ADC #$01          ;NO, SET NCONV TO MSEC10+1 TO
0F0F 85AE                 STA NCONV         ; ENSURE NO SLEWING ERROR
0F11               ;
0F11 A2DD          START  LDX #$DD          ;GET READY TO START
0F13 A940                 LDA #$40          ;SET TIMER T1 FOR
0F15 8D0BA0               STA ACR           ; CONTINUOUS OPERATION
0F18 A910                 LDA #$10          ;LOAD T1CL
0F1A 8D04A0               STA T1CL
0F1D A927                 LDA #$27          ;LOAD T1CH AND START THE
0F1F 8D05A0               STA T1CH          ; CLOCK AT 10 MSEC RATE
0F22               ;START THE EXPERIMENT USING CB2
0F22 8E0CA0               STX PCR           ;SET B/C AND CB2 LOW
0F25 A200                 LDX #$00          ;INITIALIZE DATA ARRAY POINTER
0F27 A4AE          LOOP1  LDY NCONV         ;SET UP NO. OF ADC'S/POINT
0F29 A5AD                 LDA MSEC10        ;GET NO. OF 0.01 SECS. AND
0F2B 85A9                 STA CTR           ; SET UP LOOP COUNTER
0F2D A900                 LDA #$00
0F2F 85AA                 STA BUFFLO        ;CLEAR ADC BUFFER
0F31 85AB                 STA BUFFHI
0F33 9D800F               STA DATLO,X       ; AND CURRENT DATA POINT
0F36 9D0ADA               STA DATHI,X
0F39 A9FF          LOOP2  LDA #$FF          ;RESET B/C AND CB2 HIGH
0F3B 8D0CA0               STA PCR
0F3E A9FD                 LDA #$FD          ;B/C LOW STARTS CONVERSION
0F40 8D0CA0               STA PCR
0F43 18                   CLC
0F44 A5AA                 LDA BUFFLO        ;GET LOW BYTE OF ADC BUFFER
0F46 7D800F               ADC DATLO,X       ;AND ADD TO CURRENT DATA POINT
0F49 9D800F               STA DATLO,X
0F4C A5AB                 LDA BUFFHI        ;GET HIGH BYTE OF ADC BUFFER
0F4E 7D0ADA               ADC DATHI,X       ; AND ADD TO CURRENT DATA POINT
0F51 9D0ADA               STA DATHI,X
0F54 AD00A0        WAIT   LDA IRB           ;IS THE CONVERSION DONE?
0F57 30FB                 BMI WAIT          ;NO, GO CHECK AGAIN
0F59 85AB                 STA BUFFHI        ;YES, STORE HIGH BYTE IN BUFFER
0F5B AD01A0               LDA IRA           ;GET LOW ORDER BITS AND
0F5E 85AA                 STA BUFFLO        ;STORE LOW BYTE IN BUFFER
0F60 88                   DEY
0F61 10D6                 BPL LOOP2         ;IF MORE READINGS, RESTART THE ADC.
0F63 E8                   INX               ;NO, UPDATE DATA POINTER
0F64 2C0DA0        TWAIT  BIT IFR           ;10 MSEC INTERVAL UP YET?
0F67 50FB                 BVC TWAIT         ;NO, WAIT FOR TIMER FLAG
0F69 2C04A0               BIT T1CL          ;YES, CLEAR TIMER FLAG
0F6C C6A9                 DEC CTR           ;END OF TIME ON THIS POINT?
0F6E D0F4                 BNE TWAIT         ;NO, WAIT SOME MORE
0F70 E032                 CPX #$32          ;YES, ALL 50 POINTS DONE?
0F72 D0B3                 BNE LOOP1         ;NO, GET NEXT DATA POINT
0F74 60                   RTS               ;YES, RETURN TO BASIC
                          END
```

```
100  REM  DEMONSTRATION PROGRAM FOR USE WITH SUBROUTINE DATACQ
110  DIM X(50)
120  INPUT "TIME/POINT IN UNITS OF 0.01S ";T
130  PRINT " "
140  IF T < 1 OR T > 255 THEN  STOP
150  POKE 4,0: POKE 5,15
170 I =  USR (T)
180 AL = 3968
190  REM  START ADDRESS FOR LOW BYTE OF DATA ($0F80)
200 AH = 4032
210  REM  START ADDRESS FOR HIGH BYTE OF DATA ($0FC0)
220  FOR I = 1 TO 50
230 J = AL + I - 1
240 K = AH + I - 1
250 X(I) =  PEEK (J) +  PEEK (K) * 256
260  NEXT I
270  FOR I = 1 TO 50 STEP 2
280  PRINT X(I),X(I + 1)
290  NEXT I
300  PRINT " "
310  GOTO 120
320  END
```

are those of the floating point accumulator, and as such, have no effect on the operation of BASIC.

The number of analog-to-digital conversions at each point (NCONV) is a variable depending on the time interval. In order to avoid any slewing error, NCONV is set to MSEC10 + 1, up to a maximum of 64, which is the largest number of successive 10-bit conversions that can be added together (digital smoothing) without any overflow of the 16-bit data storage format. This arbitrary process ensures 10-bit accuracy for all exponential reactions which are followed to 99% completion. All data recording is done in the first 10 msecs. with each conversion and its associated processing taking less than 60 μ sec. To speed up throughput, part of the data processing is done during the AD571 conversion time so that it is necessary to perform a dummy conversion to obtain the final reading at each point.

MICRO

Lazer isn't afraid to compare!

(because we have the best lower case system available.)

Despite the fact that we were one of the first manufacturers to produce lower case equipment for the Apple II, Lazer MicroSystems products are still the state-of-the-art. Beside the obvious price/performance advantage we have over the competition, our products are expandable. Lazer is constantly introducting new products including our Lower Case + Plus II, Character Set + Plus (that adds 2 additional character sets to the Lower Case + Plus), and our new "Double Vision + Plus" for owners of Computer STOP's Double Vision 80-column board.

Lazer's products are compatable with more word processors than anybody else's. Our Lower Case + Plus is compatable with Easywriter and unmodified Apple writers. None of our competitors below can make that claim. In fact, BASIS' board isn't even compatable with Programma's PIE! The following chart lists Lazer's superiority over the competing units.

(LC+ = Lower Case + Plus LC+II = Lower Case + Plus II KB+ = Keyboard + Plus)

Feature	Paymar LCA-1	Paymar LCA-2	VIDEX	BASIS	VISTA	LC+	LC+II	KB+/ LC+II	KB+/ LC+	KB+
True ASCII upper/lower case display	Y	Y	Y	Y	N	Y	Y	Y	Y	N
Inverse Lower Case	N	N	rev 7 only	N	—	Y	N	N	Y	—
Font Size	5 x 7	5 x 7	5 x 8	5 x 8	—	5x7, 7x8	5 x 7	5 x 7	5x7, 7x8	—
# of on-board character sets	1	1	1	1	—	up to 4 (2 std)	1	1	up to 4	—
Pseudo-descenders	Y	Y	N	N	—	Y	Y	Y	Y	—
True descenders	N	N	Y	Y	—	optional	N	N	optional	—
Optional fonts avail. (ROM, disk)	N	N	N	Y	—	Y	N	N	Y	—
2716-compatible character generator compatable with fonts created by HIRES character generators	N	N	N	N	—	Y	N	N	Y	—
On-board graphics character set	N	N	N	N	—	Y	N	N	Y	—
Software provided on diskette	$5 extra		N	N	—	Y	N	N	Y	Y
Single board works with all Apples	N	N	N	N	Y	Y	N	N	Y	Y
Expandable System	N	N	N	N	N	Y	Y	Y	Y	Y
Extensive user Documentation	N	N	Y	N	N	Y	Y	Y	Y	Y
High quality PC board	N	—	Y	Y	Y	Y	—	Y	Y	Y
Reset key disable	N	N	Y	Y	N	N	N	Y	Y	Y
Shift key mod	N	N	Y	Y	N	N	N	Y	Y	Y
All 128 characters available from keyboard	—	—	N	N	—	—	—	Y	Y	Y
Type ahead buffer	N	N	N	N	Y	N	N	Y	Y	Y
# of characters in buffer	—	—	—	—	40	—	—	64	64	64
Ability to clear or turn off buffer	—	—	—	—	N	—	—	Y	Y	Y
PRICE	59.95	49.95	129.95	125.00	49.95	64.95	29.95	129.90	164.90	99.95

Don't settle for anything less than the Lower Case + Plus, Lower Case + Plus II, or Keyboard + Plus. If your dealer doesn't stock our products give us a call, we'll try to connect you with a dealer in your area.

Before you buy a lower case system for the Apple II, call or write for our free booklet "Keyboard Enhances/Buffers and Lower Case Adapters: From The Inside Out." This booklet explains the advantages and disadvantages of using a lower case adapter in your Apple II.

Lazer products are carried by reputable dealers all across the world. If your dealer cannot provide you with a demonstration of our equipment and tries to sell you an inferior lower case adapter, give us a call, we'll give you the location of a dealer that can show you our equipment. Remember, an intelligent purchase cannot be made if you do not compare before buying.

State of the art performance, software compatability, and exceptional value make Lazer's products the best there are!

Lower Case + Plus	$64.95
Keyboard + Plus	$99.95
Lower Case + Plus II	$29.95
Character Set + Plus	$24.95
Double Vision + Plus	$39.95
Anix 1.0	$49.95
Lazer Pascal	$39.95

Visicalc Users! Now you can have lower case on your Visicalc Screen, *only* from Lazer.

Lazer MICRO SYSTEMS INC.

1791-G Capital
Corona, CA 91720
(714) 735-1041

JUDGE THE REST, THEN BUY THE BEST

Only GIMIX offers you **SOFTWARE SWITCHING** between **MICROWARE's OS-9** and **TSC's FLEX.** Plus you get the power of the GMXBUG system monitor with its advanced debugging utility, and memory manipulation routines. A wide variety of languages and other software is available for these two predominant 6809 Disk Operating Systems.

You can order a system to meet your needs, or select from the 6809 Systems featured below.

JUDGE THE FEATURES AND QUALITY OF GIMIX 6809 SYSTEMS

GIMIX' CLASSY CHASSIS™ is a heavyweight aluminum mainframe cabinet with back panel cutouts to conveniently connect your terminals, printers, drives, monitors, etc. A 3 position keyswitch lets you lock out the reset switch. The power supply features a ferro-resonant constant voltage transformer that supplies 8V at 30 amps, + 15V at 5 amps, and − 15V at 5 amps to insure against problems caused by adverse power input conditions. It supplies power for all the boards in a fully loaded system plus two 5 ¼" drives (yes! even a Winchester) that can be installed in the cabinet. The Mother board has fifteen 50 pin and eight 30 pin slots to give you the most room for expansion of any SS50 system available. 11 standard baud rates from 75 to 38.4K are provided and the I/O section has its own extended addressing to permit the maximum memory address space to be used. The 2 Mhz 6809 CPU card has both a time of day clock with battery back-up and a 6840 programmable timer. It also contains 1K RAM, 4 PROM/ROM/RAM sockets, and provides for an optional 9511A or 9512 Arithmetic Processor. The RAM boards use high speed, low power STATIC memory that is fully compatible with any DMA technique. STATIC RAM requires no refresh timing, no wait states or clock stretching, and allows fast, reliable operation. The system includes a 2 port RS232 serial interface and cables. All GIMIX boards use gold plated bus connectors and are fully socketed. GIMIX designs, manufactures, and tests in-house its complete line of products. All boards are twice tested, and burned in electrically to insure reliability and freedom from infant mortality of component parts. All systems are assembled and then retested as a system after being configured to your specific order.

56KB 2MHZ 6809 SYSTEMS WITH GMXBUX/FLEX/OS-9 SOFTWARE SELECTABLE

With #58 single density disk controller **$2988.59**
With #68 DMA double density disk controller **$3248.49**
to substitute Non-volatile CMOS RAM with battery back-up, add 300.00
for 50 Hz export power supply models, add 30.00

Either controller can be used with any combination of 5" and/or 8" drives, up to 4 drives total, have data recovery circuits (data separators), and are designed to fully meet the timing requirements of the controller I.C.s.

5 ¼" DRIVES INSTALLED IN THE ABOVE with all necessary cables

	SINGLE DENSITY		DOUBLE DENSITY		
	Formatted	Unformatted	Formatted	Unformatted	
40 track (48TPI) single sided	199,680	250,000	341,424	500,000	2 for **$700.00**
40 track (48TPI) double sided	399,360	500,000	718,848	1,000,000	2 for **900.00**
80 track (96TPI) single	404,480	500,000	728,064	1,000,000	2 for **900.00**
80 track (96TPI) double	808,960	1,000,000	1,456,128	2,000,000	2 for **1300.00**

Chart shows total capacity in Bytes for 2 drives.

Contact GIMIX for price and availability of 8" floppy disk drives and cabinets; and 5" and 8" Winchester hard disk system.

128KB 2Mhz 6809 DMA Systems for use with TSC's UNIFLEX or MICROWARES's OS-9 Level 2

(Software and drives not included) **$3798.39**
to substitute 128KB CMOS RAM with battery back-up, add 600.00
for each additional 64KB NMOS STATIC RAM board, add 639.67
for each additional 64KB CMOS STATIC RAM board, add 988.64
for 50 Hz export power supply, add 30.00

NOTE: UNIFLEX can not be used with 5" minifloppy drives.

GIMIX has a wide variety of RAM, ROM, Serial and Parallel I/O, Video, Graphics, and other SS50 bus cards that can be added now or in the future. Phone or write for more complete information and brochure.

THE SUN NEVER SETS ON GIMIX USERS

GIMIX Systems are found on every continent, except Antarctica. (Any users there? If so, please contact GIMIX so we can change this.) A representative group of GIMIX users includes: **Government Research and Scientific Organizations** in Australia, Canada, U.K., and in the U.S.; NASA, Oak Ridge, White Plains, Fermilab, Argonne, Scripps, Sloan Kettering, Los Alamos National Labs, AURA. **Universities**: Carleton, Waterloo, Royal Military College, in Canada; Trier in Germany; and in the U.S.; Stanford, SUNY, Harvard, UCSD, Mississippi, Georgia Tech. **Industrial users** in Hong Kong, Malaysia, South Africa, Germany, Sweden, and in the U.S.; GTE, Becton Dickinson, American Hoechst, Monsanto, Allied, Honeywell, Perkin Elmer, Johnson Controls, Associated Press, Aydin, Newkirk Electric, Revere Sugar, HI-G/AMS Controls, Chevron. **Computer mainframe and peripheral manufacturers**, IBM, OKI, Computer Peripherals Inc., Qume, Floating Point Systems. **Software houses**: Microware, T.S.C., Lucidata, Norpak, Talbot, Stylo Systems, AAA, HHH, Frank Hogg Labs, Epstein Associates, Softwest, Dynasoft, Research Resources U.K., Microworks, Analog Systems, Computerized Business Systems.

TO ORDER BY MAIL

SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear.

U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00.

Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S.. Our bank is the Continental Illinois National Bank of Chicago, account #73-32033. Visa or Master Charge also accepted.

GIMIX INC. reserves the right to change pricing and product specifications at any time without further notice.

are registered trademarks of GIMIX Inc.

FLEX AND Uniflex are trademarks of Technical Systems Consultants Inc. OS-9 is a trademark of Microware Inc. See their ads for other **GIMIX** compatible software.

GIMIX Systems are chosen by the Pros because of quality, reliability and features.

GIMIX inc.

The Company that delivers Quality Electronic products since 1975.

1337 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

PET "Listener"

This article dascribas an aasy-to-build davica for listening to PET cassatta tapas. In addition to detacting and corracting tape troublas, it doubles as a CB2 sound amplifler.

Louis F. Sander
and Victor H. Pitre
153 Mayer Drive
Pittsburgh, Pennsylvania 15237

Soon after acquiring my PET, I learned about CB2 sound, and bought a small Radio Shack amplifier-speaker to let me hear it. A bit of experimentation showed that the amplifer was also useful for listening to tape LOADs and SAVEs, and that listening to these processes would let me detect bad LOADs, attempts to SAVE without pressing RECORD, and misaligned heads. The amplifier quickly became my most useful PET accessory, and with it I was able to avoid tape problems of any kind, and have a lot of fun using sound in my programs.

But there were some drawbacks. For instance, my amplifier's volume was hard to keep at a comfortable level, and it used up a lot of batteries. More importantly, when I wanted to switch from CB2 to tape monitoring, I bad to open up my PET and move the input lead from one place to another.

These problems were a nuisance, so a friend and I designed a simple circuit to overcome them. This article describes an easy-to-build gadget for listening to PET's CB2 sound, and for monitoring record and playback on either tape drive. Output is adjustable from silence to a pleasingly loud level, and tbere are no batteries to replace. Input switching is automatic, and several inputs can be active at once witbout disrupting operation. The unit can be built in a few bours from common parts, at a total cost comparable to that of my original speaker-amplifier.

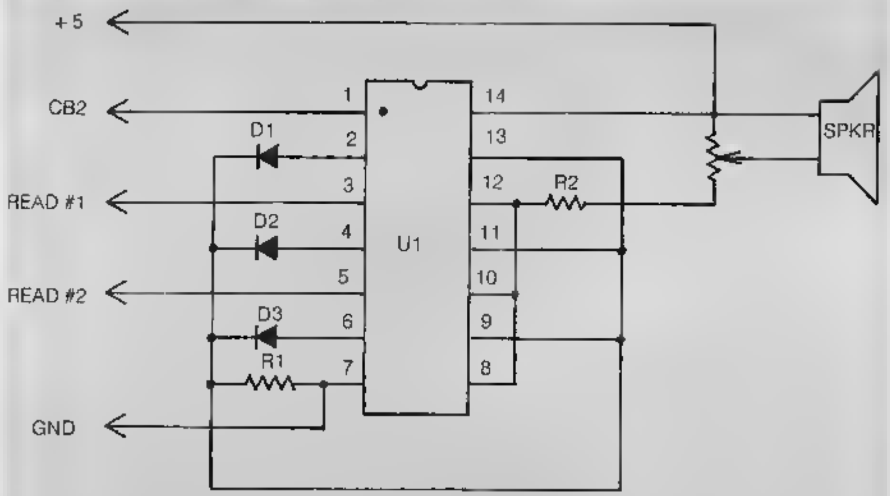
Construction

The finished Listener is pictured in figure 1. I built mine on a small piece of perfboard, which I mounted to the automobile speaker control used for adjusting volume. The perfboard and its push-in terminals make construction easy, but there is nothing critical about this method, and the unit could be built on a scrap of wood, at a savings of about half tbe total cost of parts. If you use my method of construction, mount the perfboard with flat head screws, and countersink them to give a flat bottom surface to your Listener.

Figure 2 is a schematic and wiring diagram, including a complete parts list. Just follow it as you build the Listener, and you can't go far wrong. You'll probably already have at least some of the hardware around the house. Construction is straightforward, if somewhat delicate; be careful about shorts and poor connections. Observe the proper polarity of the diodes — the end *without* the band is connected to the IC socket. Also, use pliers to protect the diodes when cutting or soldering their leads. Pin 1 of the 7404 is identified by an embossed dot, so be sure you plug it into pin I of the socket.

Figure 1: **The Listener — a useful sound device for the PET.**

***Figure 2:* Listener Construction Diagram**



Parts List

Symbol	Description	Radio Shack Part #
D1, D2, D3	1N914 Diodes	276-1122
R1	330 Ohm, ¼ Watt Resistor	271-1315
R2	47 Ohm, ¼ Watt Resistor	271-1307
R3	Auto Speaker Control (100 ohms)	40-550
SPKR	2¼'' Round Replacement Speaker	40-246
U1	Type 7404 Hex Inverter IC	276-1802
	14-Pin DIP IC Socket	276-1999
	Pre-drilled Phenolic Board	276-1395
	Push-in Terminals	270-1392
	Hookup Wire	278-1307
	6-32 × ¾'' Machine Screws (2)	64-3012
	6-32 Nuts	64-3019
	6-32 × ¼'' Spacers	64-3024
	Double-sided Foam Tape Strips	64-2344

The push-in terminals make convenient tie points for input, output and power wires, but as mentioned before, they can be dispensed with. The long wires that come on the speaker control can be cut off and used to make other connections.

Installation and Operation

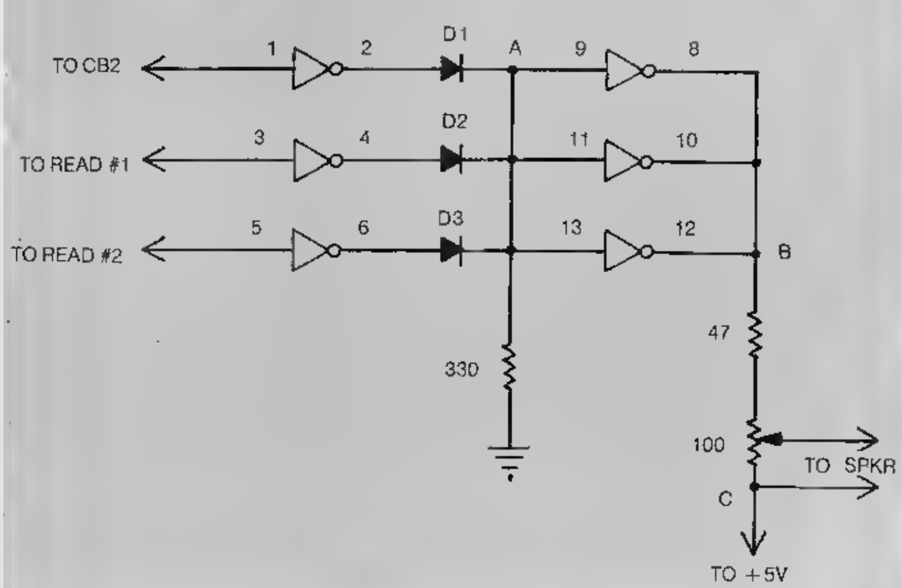
The Listener requires several connections to your PET, and the TAPE #1 connector is a good place to make most of them. I stripped tiny places on the GND, +5 and READ wires coming from my built-in recorder, and soldered the appropriate wires there. The Listener's READ #1 wire goes to the tape player's READ wire, GND goes to either one of the two GND wires, and the +5 wire goes to the +5 wire. The identities of the tape unit wires are plainly marked on PET's circuit board.

You could also make up a connector to go between the recorder and the PET, picking up these connections from it. Or you could carefully solder the wires to the proper points on the PET circuit board. If you cringe at the thought of cutting or soldering inside your machine, just poke the stripped Listener wires into the back of the tape unit connector, and rely on the mechanical connections to pick up your voltages. The READ #2 wire from the Listener goes to the READ line of tape unit #2, and the CB2 wire goes to pin M of the Parallel User Port.

Use double-sided foam tape to mount your Listener and its speaker inside the PET. I drilled a hole (shudder!) in the left side of my PET to clear the Listener shaft and give me an external volume control. If you don't like drilling, just mount the control elsewhere.

The Listener should work as soon as you power up your PET. You should hear the output from either tape whenever the cassette unit is running, whether or not PET is reading the tape. This is handy in pre-positioning tapes, and for evaluating their quality without LOADing them. In my experience, bad tapes usually *sound* bad on the Listener. In my PET, I can also hear the tape while I'm SAVEing, but this is untested with the newer recorders. If I am trying to SAVE onto a blank tape and have forgotten to press RECORD, the absence of sound alerts me to that fact. But if the same thing happens with an already recorded tape, I hear

***Figure 3:* Listener Logic Diagram**




JINSAM™ 8.0
space
age
micro
software
Used at NASA,
Kennedy Space Center
With Multiple Applications Related
to the Columbia Space Shuttle Project
DEALER INQUIRY WELCOME.
JINI MICRO-SYSTEMS, Inc.
DATABASE MANAGEMENT SYSTEM DESIGN
BOX 274 KINGSBRIDGE STN., RIVERDALE, N.Y. 10463 (212) 796-6200

the existing material, and not my SAVE, so the alert doesn't work. That's just one more reason to bulk erase your tapes before SAVEing onto them.

Of course the Listener will also reproduce any sound your CB2 line puts out. The programming aspects of CB2 sound are described in detail in the *Best of the PET Gazette* and elsewhere, so we will not go into them here. Many commerical programs use it, so you will be able to take advantage of sound even if you don't know how to program it.

Your Listener should work for years without furtber attention from you, and it should provide you with many hours of fun with CB2 sound, as well as with relief from tape-connected anxiety.

Theory of Operation

The Listener's circuit is simple, and analyzing it requires only an elementary understanding of resistors, diodes and inverters. If you have any electrical knowledge at all, hut are new to digital circuits, pages 6-11 of Radio Shack's *Engineer's Notebook* (#276-5001) contain all the information you'll need to follow our analysis. Let's start by looking at figure 3, and by tracing what happens when PET's CB2 line begins sending a tone. If the tone happens to be 1000 Hz, CB2 switches from high to low and hack 1000 times each second. The loudspeaker follows the switching, and we hear the tone. This is why we hear it:

Consider the Listener's CB2 input, which is connected to pin 1 in figure 3. Before the tone begins, CB2 and both other inputs are idling at a high logic level of +5 volts. The inverter between pins 1 and 2, seeing a high level at pin 1, puts a low logic level, or zero volts, onto pin 2. D1 has no effect in this case, so point A and pin 9 are also low. Another inverter, actually three in parallel, makes pin 8 high. Since there is +5 volts at points B and C, there is no voltage drop between them, and the speaker sees no voltage across its voice coil. Evcrything is silent.

When CB2 begins sending its tone, pin 1 goes to a low level, and the two inverters take pin 8 low, too. This puts 5 volts across B and C (0 volts at B, +5 volts at C). The speaker sees some amount of this voltage, depending on the setting of the volume control. At the end of the tone's first pulse, CB2 goes high again, taking pin 8 high, and removing tbe voltage drop across the speaker. This hack and forth routine continues as long as CB2 keeps sending its pulses. The speaker follows it, and we hear the tone.

If either tape begins to play, its READ pins begin switching from high to low at an audio rate, and the resulting sound gets through to the speaker in exactly the same way as the CB2 tone. If several inputs are active at one time, the speaker will follow all of them, and there will be no adverse interaction except in your own ears, because D1 - D3 isolate everything to their left from the voltage swings at point A. The 330 ohm resistor 'pulls down' point A to zero volts as soon as pins 2, 4 and 6 go low, and that's all there is to the circuit.

Now that you know how it works, doesn't that CB2 music sound just a *little* sweeter?

Editor's Note: Programming CB2 Sound

The author follows Hal Chamberlain's convention of using pins M (CB2) and N (ground) from the parallel user port.

Three addresses are involved:

POKE 59467,16 (sound on) or 0 (sound off)

POKE 59464,1 - 255 (high to low pitch)

POKE 59466,15 or 51 or 85 (three different ranges)

When done with sound *always*:

POKE 59467,0 : POKE 59464,0 : POKE 59466,0

If you don't do this, your cassette won't work properly!

MICRO

Watch that Ground Connection

A properly connected ground is more then e preceution egeinst shock — even the smellest oversights cen leed to eerious demege to your computer'e components.

Raymond Weisling
Jalan Citropuran No. 23
Surakarta, Jawa Tengah,
Indonesia

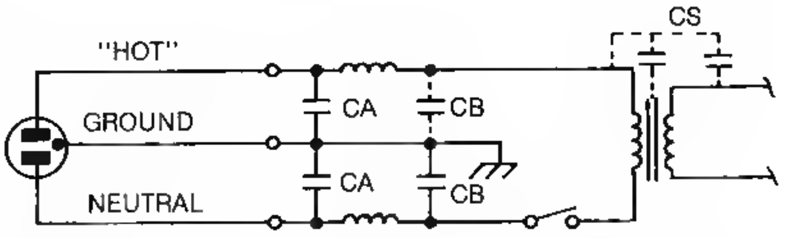


Figure 1: **Typical Power Supply Primary (Fuse and Pilot Lamp Omitted)**

As typical computer systems grow in size and complexity, with more and more pieces of equipment interconnected, there is a growing danger of damage to sensitive circuits from casual interconnect practices. Here we will look at some of these dangers, their causes, and what protective measures can be taken to insure safety to our expensive equipment. These dangerous practices are even more likely to strike the experimenter who uses less integrated systems; i.e., those systems which are huilt around smaller, less packaged devices such as the single hoard computer and its peripherals.

Exactly what dangers are we talking about? The source of most of the prohlems, or potential problems, is the mains power line that supplies the 115/230 volt AC power. For human safety, the United States (and other countries as well) has moved toward adopting a three-conductor plug-socket standard, where the third wire is an earth ground connection. The major idea here is that cases and frames of appliances can he assured of a good ground, in the event of an internal short to the case or frame, preventing a potentially fatal situation. Most of us are aware of tbis, hut at the same time, most of us are also aware of widespread misuse of the intended safety feature. Many homes are not equipped with the newer sockets, and so the ground pin is defeated in some way. (The U.S. style plug is more easily abused in this way than some other styles in use in the world.) We go on using the equipment, whicb works just as well without this ground connection. Barring the rare case of a line-to-case short, there is no prohlem.

Or is there? Well, if the equipment in use is a computer-related device, serious damage can result due to misuse of ground connections. The same damage can even occur if there is such a ground connection, hut where it has failed to make proper contact. (Worn sockets or hroken wires inside the cable are typical causes.) Let us analyze the prohlem to understand how this can happen.

All of the computer devices, printers, disk systems, CRT's, etc., have power transformers, and many now employ line filters (see figure 1). The transformers usually bave some capacitance hetween the primary and the iron core and the secondary. This represents a leakage path for the AC power. The use of a noise filter is guaranteed to offer a path for the AC line to the ground, or frame. Figure 2 shows some values of such capacitance and the possihle current that can flow into the ground. Note that the noise filter configuration is a voltage divider for this AC flow, since the neutral line is usually well-connected (or else the equipment cannot operate), and thus the current availahle is half as great as in figure 2. But if the mains power is 230 volts, the current will he doubled.

If one of a group of devices has an open ground line, while the others are safely grounded, and if the data connector is inserted or removed, this cur-

Total Capacitance CA+CB+CS	Reactance At 60 Hz	Current In Ground At 115V
100 pf	25 M Ω	4 μA
1.0 nf	2.5 M Ω	40 μA
10 nf	250K Ω	400 μA
100 nf	25K Ω	4 mA
1.0 μf	2.5K Ω	40 mA

Figure 2

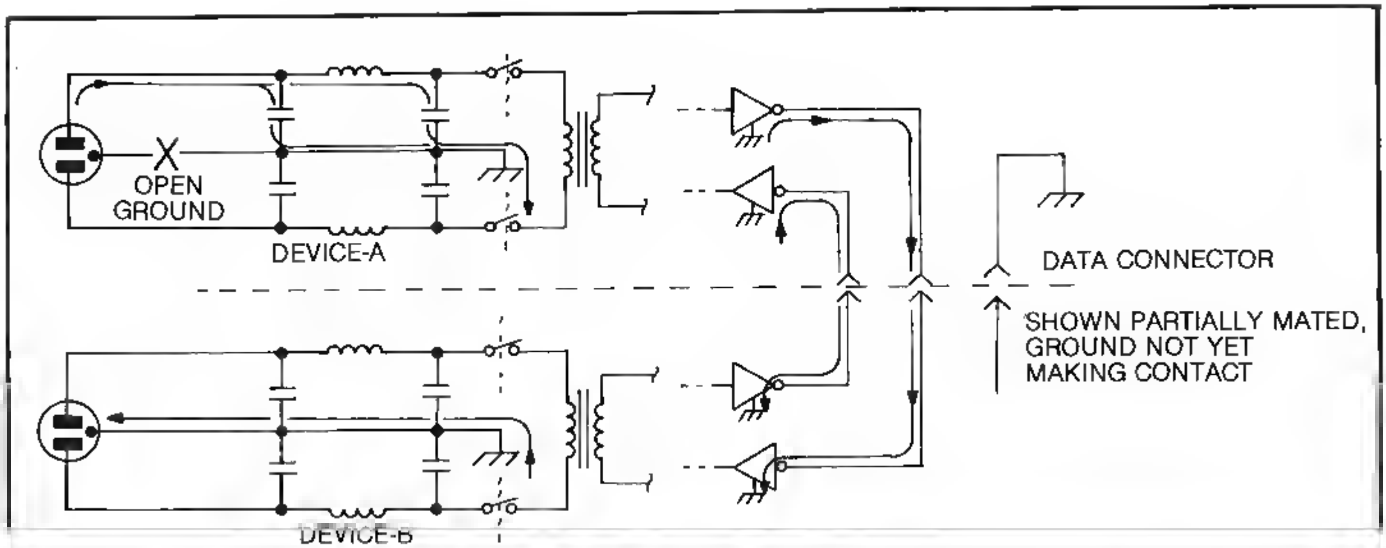


Figure 3: **Death of a chip. Arrows show current path from Device-A to Device-B, *via* I.C. devices, despite both power switches being "off."**

rent can flow along an *unpredictable* path between the two pieces of equipment. While the items are connected, there is no problem, since the data ground path also carries the power line leakage current. But when the connectors are partly connected, as during insertion or withdrawal (or even an inadvertant withdrawal where no locking mechanism is in use), the data ground may not be connected while other data lines are connected. This is the kiss of death. (See figure 3.) The sensitive input circuits, or even the output drivers, may not take this abuse. Any on-chip protection diodes may not handle the current since they are small structures; off-chip protection diodes of more substantial size are not commonly employed, as they add cost and have large capacitances which degrade the risetimes of the data signals. And, since the density of ICs is increasing while the power dissipation is decreasing, the newer devices are more sensitive to such abusive voltage and current. An important thing to remember is that this danger does not disappear when the equipment is turned off, since the input line filters are usually located upstream of the mains switch. Further, if only a single pole switch is employed, there is a chance that the *hot* side of the line is still connected to the transformer and the capacitive-coupled path remains present.

It might be appropriate to digress here on connector design and pin assignment, if only as a reminder to designers, in view of the dangers described above. There are a few types of connectors in use where the intended ground pin is the first to make and the last to break during mating. The three-prong U.S. standard AC plug and the "Cannon XLR" audio connectors are two examples; both are equipped with this feature for different reasons. (The XLR connector used in professional audio systems can be mated even if on a "hot" microphone channel without any induced hum; the common "RCA Phono" plug used in consumer audio connections is quite the opposite, ensuring that the shield makes last.) Data connectors, on the other hand, never employ this strategy, and so whatever pin happens to make first is the one to carry any unwanted current. Equipment designers could offer a partial measure of safety by assigning the outermost pins on either side to ground. Then if the connector is accidentally partly tugged out of the socket by a taut cable, a ground on one of the two sides still makes contact. However, for straight-in manual insertion, it is still a gamble as to what will happen.

We can ensure the safety of our expensive equipment through some preventive measures. One is to be sure all equipment has a good plug and cable connection for the ground or frame. Figure 4 illustrates a simple test set for this, intended to be built into a small case and employed periodically to insure that the cable is still good. (Plug in the mains cable and clip the wire to the frame, then flex the cable, especially near either end. If the path opens, the audio tone will come on.)

Another technique is to interconnect data cables only with the equipment unplugged fully. This can be a nuisance, but if you are using a power distribution strip with only one wall plug, it is much easier. If this strip has a

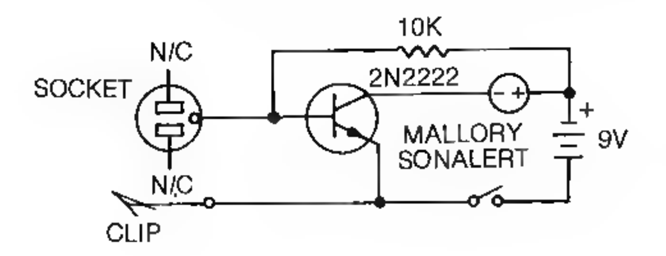


Figure 4

switch, it may be used *if* it breaks both sides of the line. (Check it with an ohmmeter.) Of course, be sure that the wall socket ground pin really is connected to a solid earth ground.

Finally, if an additional level of safety is desired, a separate ground wire can be connected to each piece of equipment, and then brought to a single common point.

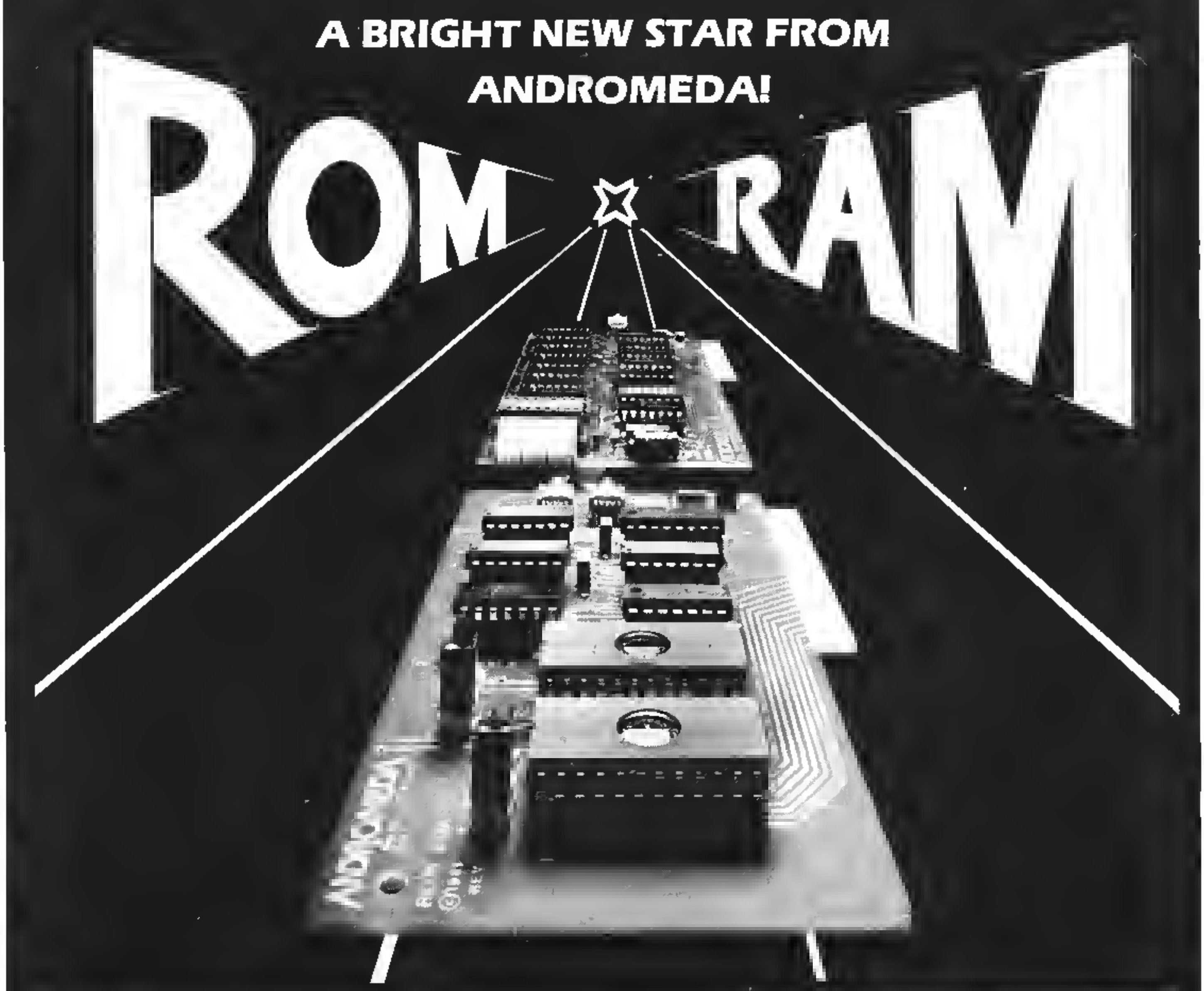
Case History

A few years ago I was programming on a minicomputer-based music synthesizer. The studio was too small for the Centronics 102A printer to be left permanently in place, so it was wheeled in each time a programming session took place. It was then connected to the computer interface wire-wrap panel and plugged into an empty wall socket. However, the whole computer system, synthesizer, 8-track tape deck, etc., was "floating" from an earth ground connection. (It is anybody's guess how much current was available from all the stray capacitances in parallel.) If the data connection was done first, there was no problem, but if the wall plug went in first, it was Russian Roulette at the data connector. Most of the time we were lucky. The first time we lost, two TTL chips in the printer died but the cause was rather a mystery. The second time more chips died, and only then did the cause become apparent. After the system was grounded there were no more problems, but since many people used that studio, I was always especially careful to plug the printer into the wall last. When you've spent six or more hours troubleshooting slain chips, you become less casual about such things.

Conclusion

Ground connections at the power socket, intended as an element of insurance against a rare, but potentially fatal (to people), short within equipment, are frequently abused or defeated for reasons of convenience. However, in the computer environment such bad practices can lead to equipment damage from AC power leakage current that comes, ironically, from other safety devices (noise filters) or from the power transformer itself. Careless or casual interconnection of equipment without consideration to this danger can cause msyterious component failure. Observing a few simple rules and ensuring that the hardware is in good condition will prevent these kinds of accidents.

MICRO



NEW ROM BOARD FOR THE APPLE II*
$125.00 WITH UTILITY ROM.

With Andromeda's new ROM Board, you can plug many useful utility programs into your Apple II. Because ROM memory never forgets, you can access these utilities instantly without having to load them from disk.

The ROM Board comes with the utility ROM, which gives you five powerful options to apply to your Applesoft* programs. With the Utility ROM, you can do automatic line numbering, control a program list with a page mode, restore a crashed Applesoft* program in memory, alphabetize a disk catalogue and create a disk without DOS, giving you an extra 8K on your disk. Any of Soft Control Systems' other ROMS can be used, such as the Dual DOS in ROM, and Your'ple ROM.

You can install 2K PROMS, 4K PROMS, or even 2K RAM chips in each of the two memory sockets. So you can even have the Read - Write capability of RAM to develop PROM Programs yourself, or just have an extra 2K RAM for your machine - Language programs. Two 2732 PROMS allow a total of 8K of memory on the Board.

Now with One Year Warranty.

ANDROMEDA
INCORPORATED
Greensboro, NC. 27410
P.O. Box 19144
919 852-1482

Price for Andromeda 16K RAM expansion board now only $120.00. Please add $5 for shipping and handling. North Carolina residents add 4% sales tax.

*DEALER INQUIRIES WELCOME.

we beat the price...

ATARI

800™ $749

ATARI Computers for people.™

ATARI 810 DISC DRIVE

$444

ATARI SOFTWARE

CX404 Word Processor	$119.00
CX405 PILOT	$68.00
CX413 MICROSOFT BASIC	$68.00
CX4101 Invitation to Programming 1	$17.00
CX4102 Kingdom	$13.00
CX4103 Statistics	$17.00
CX4104 Mailing List	$17.00
CX4105 Blackjack	$13.00
CX4106 Invitation to Programming 2	$20.00
CX4107 Biorhythm	$13.00
CX4108 Hangman	$13.00
CX4109 Graph It	$17.00
CX4110 Touch Typing	$20.00
CX4111 Space Invaders	$17.00
CX4112 States & Capitals	$13.00
CX4114 European Countries & Capitals	$13.00
CX4115 Mortgage & Loan Analysis	$13.00
CX4116 Personal Fitness Prog	$59.00
CX4117 Invitation to Programming 3	$20.00
CX4118 Conversational French	$45.00
CX4119 Conversational German	$45.00
CX4120 Conversational Spanish	$45.00
CX4121 Energy Czar	$13.00
[illegible]	$45.00
CX6001 U.S. History	$23.00
CX6002 U.S. Government	$23.00
CX6003 Supervisory Skills	$23.00
CX6004 World History	$23.00
CX6005 Basic Sociology	$23.00
CX6006 Counseling Proced	$23.00
CX6007 Principal of Act	$23.00
CX6008 Physics	$23.00
CX6009 Great Classics	$23.00
CX6010 Business Comm	$23.00
CX6011 Basic Psychology	$23.00
CX6012 Effective Writing	$23.00
CX6014 Principals of Econ	$23.00
CX6015 Spelling	$23.00
CX6016 Basic Electricity	$23.00
CX6017 Basic Algebra	$23.00
CX8106 Bond Analysis	$20.00
CX8107 Stock Analysis	$20.00
CX8108 Stock Charting	$20.00
CXL4001 Education System Master	$21.00
CXL4002 Basic Computing Language	$46.00
CXL4003 Assembler Editor	$46.00
CXL4004 Basketball	$24.00
CXL4005 Video Easel	$24.00
CXL4006 Super Breakout	$30.00
CXL4007 Music Composer	$45.00
CXL4009 Chess	$30.00
CXL4010 3-D Tic-Tac-Toe	[illegible]
CXL4011 Star Raiders	$39.00
CXL4012 Missle Command	$32.00
CXL4013 Asteroids	$32.00
CXL4015 TeleLink	$20.00
Compuhome	$74.95
Visicalc	$149.00
Letter Perfect (Word Processor)	$119.00
Source	$89.00

Atari® Peripherals:

400 16K	$329.00
410 Recorder	$59.00
822 Printer	$359.00
825 Printer	$ CALL
830 Modem	$159.00
850 Interface	$ CALL

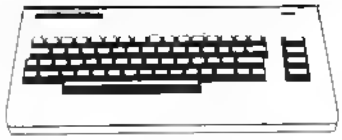
Atari® Accessories

New DOS 2 System	$21.00
CX70 Light Pen	$64.00
CX30 Paddle	$18.00
CX40 Joy Stick	$18.00
CX853 16K RAM	$89.00
Microtek 16K RAM	$75.00
[illegible]	[illegible]

CBM 8032 $1099

4016	$799.00
4032	$999.99
8096	$1795.00
CBM4022 Printer	$629.00
Tally 8024	$1699.00
CBM C2N Cassette Drive	$69.00
CBM4040 Dual Disk Drive	$999.00
CBM8050 Dual Disk Drive	$1349.00

commodore



VIC 20 $259

Vic-TV Modual	$19.00
Vic Cassette	$69.00
Vic Disk Drive	$ Call
Vic 6 Pack program	$44.00

CBM Software

WordPro3 Plus	$199.00
WordPro4 Plus	$299.00
Commodore Tax Package	$399.00
Visicalc	$149.00
BPI General Ledger	$329.00
OZZ Information System	$329.00
Dow Jones Portfolio	$129.00
Pascal	$239.00
Legal Time Accounting	$449.00
World Craft 80	$289.00
Word Check	$180.00
Create-A-Base	$239.00
Power	$89.00
Socket-2 Me	$20.00
Jinsam	Call

Disks

CX8100 Blank Disk (5)	$22.00
Sycom Blank Disk (10)	$29.00
Maxell Blank Disk (10)	$36.00
Maxell Blank Disk (10)	$46.00

Printers

Call for prices on the new NEC models.

Epson MX-70	Call for Prices
Epson MX-80	
Epson MX-80 FT	
Diablo 630	
TEC 1500 Starwriter 25cps	$1495.00
TEC 1500 Starwriter 45cps	$1795.00

No Risk, No Deposit On Phone Orders, COD or Credit Card, Shipped Same Day You Call *

* on all in stock units

IN PA, CALL (717) 327-9575

(800) 233-8950

To Order:

Phone orders invited (800 number is for order desk only). Or send check or money order and receive free shipping. Pennsylvania residents add 6% sales tax. Add 3% for Visa or M.C. Equipment is subject to price change and availability without notice. Please call between 11 AM & 6 PM.

COMPUTER MAIL ORDER
501 E. 3RD ST., WILLIAMSPORT, PA 17701

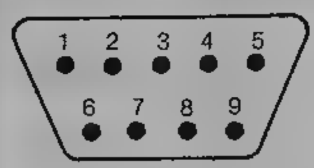
Proportional Joystick for Atari

The Ateri 800 comae aquipped with aight epacially designed A/D convartars for use with the peddias. Thaaa A/D converters mey be uead to interface to en inexpensiva $5 proportionel joystick from Redio Sheck, ellowing e much finer control for gemes and other epplicetione.

Mike Dougherty
7659 West Fremont Ave.
Littleton, Colorado 80123

The normal Atari joystick is a very simple device — built to be rugged, yet inexpensive. The joystick consists of four open circuits in each of the up, down, right, and left directions. As the stick is moved in any single direction, the appropriate circuit is closed. A diagonal move forms a combination of two adjacent closed circuits.

The joystick inputs are brought into the Atari through two 6520 PIA input ports, the values heing placed in the appropriate shadow registers hy the operating system. An open circuit appears as a logic 1 while a closed circuit appears as a logic 0. Thus, no contacts closed (the joystick not moved) is interpreted as a 15 (binary 0000 1111), DOWN is a 13 (hinary 0000 1101), UP is a 14 (binary 0000 1110), RIGHT is a 7 (hinary 0000 0111), and LEFT is an 11 (hinary 0000 1011). This method of control reduces the angular resolution of the joystick direction to 45°. (0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315° are the only allowed angles.) In addition, there is no method to indicate "how much" — the joystick is either pulled in a specific direction or it is not.



1) Joystick UP
2) Joystick DOWN
3) Joystick LEFT
4) Joystick RIGHT
5) B Paddle
6) Joystick Trigger
7) +5 Volts
8) Ground
9) A Paddle

Figure 1: **Controller Jack Pin Asaignment**

This binary approach is similar to driving a vehicle that can only move at 55 mph or he parked — there is no way to "accelerate" or "slow down." To some degree, the prohlems created hy this particular joystick design can he overcome by proper software.

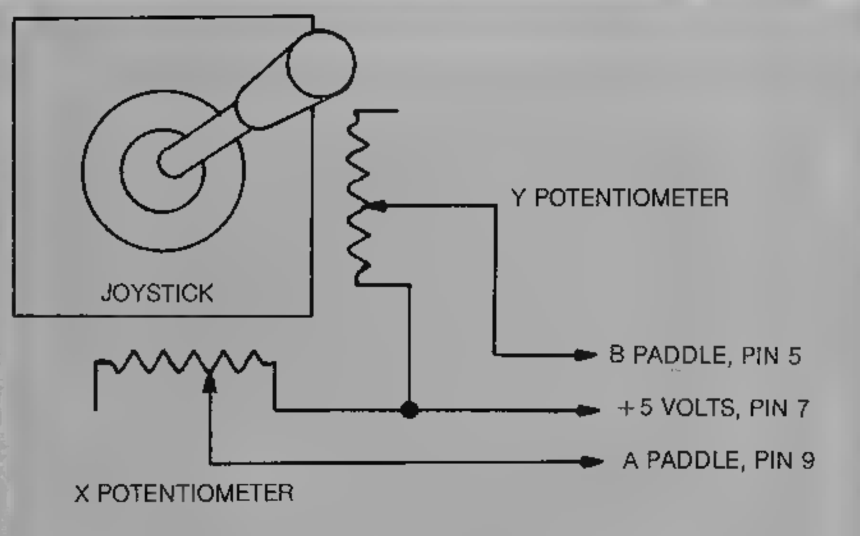


Figure 2: **Joystick Interface**

A smooth operating joystick is available from Radio Shack, catalog #271-1705, for about $5. This proportional joystick has a 100 KΩ linear potentiometer for both the X and Y directions. The total movement is 30 ± 3 degrees in each direction. In other words, the resistance in the X and Y potentiometers is directly proportional to the X, Y position of the joystick with a range of nearly 0 Ω to 100 KΩ. This joystick, when mounted in a suitable case, has all the physical characteristics required of an operably smooth joystick.

Fortunately, the problem of interfacing the joystick to the Atari 800 has already been solved. Each controller jack input contains the four normal Atari joystick inputs (closed/open circuits for each direction), the joystick trigger (also a closed/open circuit), two paddle inputs, a 5-volt source and a ground (see figure 1). Upon the examination of a paddle controller, this controller turned out to be nothing more than a 1 MΩ potentiometer. The paddle input circuit digitizes the resistance (voltage) at the A or B paddle input, while the operating system places this value in the paddle shadow registers each 1/60th of a second. The values sampled range from 1 (less than approximately 1200 Ω) to 228 (greater than approximately 700 KΩ). Thus, to digitize an external potentiometer value, the 5 volts of the controller jack should be applied to one of the two potentiometer inputs, and the potentiometer wiper output (middle connector) wired to either the A or B paddle input (depending on which PADDLE(n) is used by the software). For potentiometers with a range less than 1 MΩ, the digitized value will simply be less than 228.

The potentiometers in the Radio Shack joystick have a resistance range of 100 KΩ, giving an Atari paddle range of 1 to 42. For the purpose of a joystick, this is more than enough resolution. Listing 1 demonstrates a simple method to use the joystick to control a graphic dot on the screen. Figure 2 shows the details of the joystick interface. The joystick input is normalized in the program by a user-chosen value, 1 to 42, to reduce the step size taken each time in the main loop. Program 1 moves the dot proportional to the value of the joystick paddle inputs, allowing the user to accelerate and decelerate the dot as desired. The trace mode allows an an elaborate version of an "Etch-a-sketch." This program demonstrates how the joystick input would be used to affect objects in an application program.

The Atari joystick is adjusted to return to the center position when not being used. However, the Radio Shack joystick has no such provision. One solution is to ignore the joystick input when the values drop below a certain threshold level. This solution creates a "dead" area around the center position, allowing for imperfect human

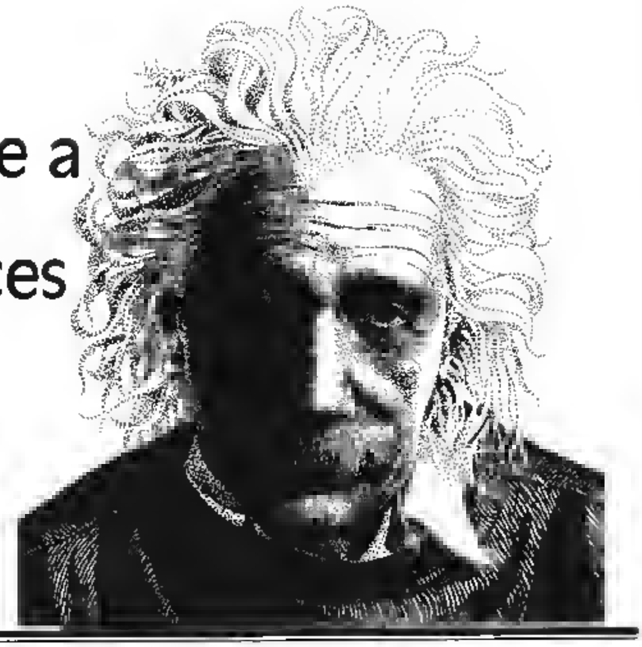
```
1 REM JOYSTICK BY
2 REM Mike Dougherty
3 REM
4 REM USING THE PADDLE A/D INPUT TO
5 REM IMPLIMENT AN INEXPENSIVE PRO-
6 REM PORTIONAL JOYSTICK.
7 REM
8 REM ............................
9 REM
10 DIM ANSWER$(1)
20 GRAPHICS 0
60 POSITION 10,5:PRINT "Trace Mode (Y/N)  ";:INPUT ANSWER$
70 POSITION 10,7:PRINT "Step scale (1-42) ";:INPUT SCALE
80 POSITION 10,9:PRINT "Threshold  (0-42) ";:INPUT THRESH
100 REM
101 REM ..SET UP DRAWING FIELD
102 REM
110 GRAPHICS 8+16:REM HIGH RES
115 SETCOLOR 2,7,0:REM SET TO YOUR OWN FAVORITE COLOR
120 COLOR 1
130 X=160:Y=95:REM STARTING PLACE
140 A=0:REM X INPUT PADDLE CHANNEL
150 B=1:REM Y INPUT PADDLE CHANNEL
200 REM
201 REM ..MAIN LOOP:
202 REM ....SAMPLE THE PADDLES
203 REM ....IF CENTERED, GIVE AUDIO FEEDBACK
204 REM ....COMPUTE NEW POSITION AND ADJUST FOR SCREEN LIMITS
205 REM ....IF NOT TRACING, ERASE OLD POINT
206 REM ....PLOT NEW POSITION
207 REM ....IF A SPACE IS PRESSED, WAIT UNTIL ANOTHER KEY IS PRESSED
208 REM ..CONTINUE LOOP
209 REM
210 REM
211 REM ..INPUT JOYSTICK THRU PADDLES
212 REM
215 XDELT=-INT((PADDLE(A)-22)/SCALE)
220 YDELT=-INT((PADDLE(B)-22)/SCALE)
225 REM
226 REM
227 REM ..CHECK FOR EXTENDED JOYSTICK
228 REM .."DEAD" CENTER POSITION
229 REM
230 IF (ABS(XDELT)<THRESH) AND (ABS(YDELT)<THRESH)THEN XDELT=0:YDELT=0
235 REM
236 REM
237 REM ..AUDIO FEEDBACK FOR CENTER POSITION
238 REM
240 SOUND 0,0,0,0:IF (XDELT=0) AND (YDELT=0) THEN SOUND 0,120,10,2
245 REM
246 REM
247 REM ..NEW POSITION BASED UPON PROPORTIONAL
248 REM ..JOYSTICK VALUE -- KEEP ON SCREEN
249 REM
250 XNEW=X+XDELT
260 YNEW=Y+YDELT
270 IF XNEW<1 THEN XNEW=1
280 IF YNEW<1 THEN YNEW=1
290 IF XNEW>318 THEN XNEW=318
300 IF YNEW>188 THEN YNEW=188
305 REM
306 REM
307 REM ..ERASE OLD POINT IF NOT IN TRACE MODE
309 REM
310 COLOR 0
320 IF ANSWER$="N" THEN PLOT X,Y:PLOT X+1,Y+1:PLOT X-1,Y-1:PLOT X+1,
    ,Y+1 Y-1:PLOT X-1
325 REM
326 REM
327 REM ..PLOT CURRENT DOT POSITION
328 REM
330 COLOR 1
340 X=XNEW
350 Y=YNEW
360 PLOT X,Y:PLOT X+1,Y+1:PLOT X-1,Y-1:PLOT X+1,Y-1:PLOT X-1,Y+1
364 REM
365 REM
366 REM ..IF A SPACE IS PRESSED THEN
367 REM ..WAIT FOR ANOTHER KEY.
368 REM ..REPEAT LOOP
369 REM
370 IF PEEK(764)=33 THEN GOTO 370
380 GOTO 210
```

judgement. A second solution is to connect one of the joystick inputs through a momentary switch to the ground in the controller jack. The software could be written to use the joystick paddle values only when this specific switch is pressed (the circuit is closed and the corresponding bit is zero). Thus, when the dot has been moved to the proper position, simply let go of the momentary switch. These and other solutions each have strong points suited for specific applications.

As an expansion to this simple project, recall that the joystick inputs and the joystick trigger are simple open/closed circuits. Thus, with five momentary contact switches, the Radio Shack joystick, a suitable enclosure, and a nine-pin "D" connector, a high quality control system may be built to run from a single controller jack.

MICRO

"You don't have to be a genius to figure out that the lowest prices on computers and peripherals are at **The Place Where You Go To Buy Computers, Inc.**"



apple computer
Authorized Dealer

APPLE II PLUS	APPLE WRITER	EPSON
APPLE III	GRAPHICS TABLET	AMDEX
DOS 3.3	PEACHTREE	ZENITH
APPLE PLOT	DC HAYES ASSOC.	THE SOURCE
APPLE PASCAL	NEC	VISICALC
APPLE FORTRAN	MICROSOFT	VISITREND

AUTHORIZED DEALER AND SERVICE CENTER

MAIL ORDER SPECIALS

QTY.	DESCRIPTION		TOTAL
____	EPSON MX 80	$444.99	____
____	EPSON MX 80 F/T	$574.95	____
____	EPSON MX 100 F/T	$724.95	____
____	EPSON INTERFACE	$ 79.99	____
____	NUMERIC 24 KEY MULTIFUNCTION PAD	$139.95	____
____	NEC GREEN MONITOR 12"	$219.95	____
____	AMDEX COLOR MONITOR 13"	$349.99	____
____	ZENITH GREEN SCREEN 12"	$124.95	____
____	COOL STACK W/FAN	$ 59.95	____
____	COOL STACK WITHOUT FAN	$ 32.95	____
____	MICROSOFT RAM CARD	$159.95	____
____	MICROSOFT Z-80 CARD	$284.99	____
____	GRAPPLER INTERFACE	$159.99	____
____	GRAFTRAX 80	$ 89.99	____
____	SUP "R" TERM	$319.95	____
____	VIDEX	$279.95	____
____	Apple Sider™	$ 29.95	____
		Sub-Total	____
		Calif. Residents Add 6% Sales Tax	____
		Add 4% for shipping & handling TOTAL	____

TO ORDER CALL (213) 78-THINK

Name ____________________
Address ____________________
City, State, Zip ____________________
Phone ____________________
Credit Card # ____________________
□ VISA □ MASTERCARD Exp. Date ____________________
Signature ____________________

Just In Time For Christmas!!

Apple Sider™

Apple Sider is the only device thest makes the outside of your apple as usetul es the inside.

This versatile little unit, which plugs into the game I/O and attaches to the side of your APPLE allows you to operate in one of two modes. The first mode enables you to easily select between one of two devices that plugs into your computer (for example joysticks and paddles). You will never have to open your APPLE again to change game devices. The second mode enables you to take advantage of your APPLE'S ability to handle up to four game controllers at one time. In either mode Apple Sider™ can be configured to meet your specific needs, such as exchanging the X for the Y axis or changing push button numbers.

Apple Sider™ makes a great stocking stuffer, and is available exclusively from, **"THE PLACE WHERE YOU GO TO BUY COMPUTERS, INC."**

ORDER NOW

Only **$29.95**

the place where you go to buy **Computers** incorporated

"the complete communications and computer store"

5848 Sepulveda Blvd., Van Nuys, CA 91411
(213) 78-THINK

APPLE II, APPLE III, APPLE WRITER, APPLE PLOT, APPLE PASCAL & APPLE FORTRAN ARE ALL REGISTERED TRADEMARKS OF APPLE COMPUTER INC.

MICRO Letterbox

Apple, OSI Readers Speak Out

Dear Editor:

I am an Apple user who loves to read computer journals — obviously about the Apple.

I came very close to cancelling my subscription to MICRO until your magazine started the Apple Bonus section. The last two or three issues have been very good and I decided to continue my subscription.

I believe it is important to provide the majority of your readers with articles that they can use, and it is apparent that Apple users are in the majority (evidence from your poll). I think it is a grave error to try to cover too much ground because, in trying to please everyone, you may be able to satisfy no one!

Please keep the Apple articles coming. I wish you continued success.

Warren Ostlund, M.D.
6616 Southcrest Drive
Edina, MN 55435

Dear Editor:

MICRO #40 showed that 39% of your readers have OSI systems. I hope this will cause an increase in the numbers of articles written for OSI. It would be nice to see 39% of the systems-oriented articles for OSI; after all, we are paying for 39% of your (our?) magazine.

Dennis W. Smith
557 S. 10th
Salina, KS 67401

Editor's note: The types of articles we publish are directly related to the material we receive from our authors. Recently we've been inundated with Apple articles, but have received little OSI material. If you're an OSI user, why not try submitting a program to MICRO that you've developed on your system! We are also beginning to generalize articles so that a program can run on more than one system.

Dear Editor:

I have been trying to get more OSI users to flood you with articles in an effort to prevent an Apple takeover. Michel Piot was afraid his English was not good enough for MICRO, but I convinced him to send in the article anyway. I'm glad to see it appeared in the July issue (38:79).

I would like to see more articles of the type "How to convert your KIM into a Dedicated Coffee Percolator" as described in MICRO 36:16. A large fraction of your articles are being provided by us skinflint bare-board hackers.

I find the 6502 bibliography next to useless. This may have been appropriate when 6502 articles were few and far between. And listing the contents of MICRO seems redundant. Perhaps the listings should be limited to articles specifically about the 6502 chip rather than just machines using that chip.

Earl Morris
3200 Washington
Midland, MI 48640

Editor's note: We appreciate your efforts to supply us with OSI material and authors — please continue!

We have been working with Dr. Dial on shortening the Bibliography, but feel it is still a worthwhile department. Dr. Dial now includes only the most pertinent 6502 articles.

Dear Editor:

Well, you finally did it: squeezed OSI out of the September, 1981 issue entirely. But it wasn't hard to see it coming with "Challenges" lasting only four issues and the "Small Systems Journal" going next. Not your fault you say, but no attempt at a replacement.

I have stopped subscribing to better magazines than yours because they let me down, and I'm sure the OSI advertisers in the September, 1981 issue feel the same way. I bet you're not surprised that OSI is gone from the back cover.

Why not just change your name to "6809 Apple Butter" and be done with it?

William F. Hertel
P.O. Box 1226
Bullhead City, AZ 86430

Editor's note: Our September issue did contain an OSI article — "The Disk Switch," (40:15). We've scheduled an OSI feature for March 1982. (We also had an OSI feature in July 1981.) OSI users have not been forgotten!

Atari Ad Attacked

Dear Editor:

As a 6502 expert and student of intellectual property law, I resent the implications of the Atari advertisement on page 17 of your October issue, and gladly take this opportunity to set your readers straight.

Atari may not be happy with the fact that others have "adapted" their ideas to other games or computers. Tough!

Unless Atari has a *patent*, they have no complaint about "adapting" or other use of their ideas. Copyright does not afford that protection. If Atari wants the law to be otherwise, let them appeal to Congress rather than attempt to deceive your readers. I gladly announce to you and the "Patent Counsel" of Atari that I am freely adapting their ideas in programs, and will continue to do so until they gain control. Suit will have to be brought in Federal District Court of Austin, TX.

Jim Kirby
502 South Park Dr.
Austin, TX 78704

MICRO

MICRO
PET Vet

By Loren Wright

Compiled vs. Interpreted

BASIC is an interpreted language. Your BASIC program is analyzed by a program called an interpreter, which occupies most of your PET's BASIC ROMs. As each instruction in the BASIC program is encountered, it must first be recognized and then executed using a particular prepackaged routine to perform that function. It doesn't matter how many times that instruction has been encountered before; it must still be interpreted before it is executed. You can see that a lot of time gets wasted in this redundancy. Also, it means that the BASIC program must be in memory in order to be interpreted.

Compiled languages require an additional step before you can run your program. The source statements must be reduced to executable machine code by a program called a compiler. Then this reduced program may be executed directly. The source and the compiler are now dispensable, and the memory they occupied during compilation is available for other uses. However, if you need to make changes, you must go back to the source program, make the necessary changes, and recompile before you can execute the new version.

Microcomputer implementations of Pascal take an "in between" approach. The source statements are compiled to a reduced form called P-code ("P" for pseudo). This P-code is then interpreted by the "P-machine," (which is really another program). This can operate faster and more efficiently because of the reduced form of the program. The P-code itself is not directly executable by the 6502. In both purely compiled and P-code languages, the source program does not have to be in memory when execution takes place.

Alternate Languages for the PET

It seems we have been deluged in the past year by alternative languages for the PET, all purporting to be better than BASIC in one or more important ways. Currently on the market are compiled BASICs, the extended Waterloo MicroBASIC (interpreted) for the SuperPET, at least two Pascal versions, and several versions of FORTH. In addition, there are non-standard languages which combine the features, advantages, and disadvantages of the better-known languages, and add some of their own.

Next month we'll focus on Pascal. The three-month "Pascal Tutorial" series by Victor Fricke concludes, and will be accompanied by several other articles to help you learn more about Pascal, its inner workings, and some applications. The PET Vet column will survey the available Pascals for the PET. The February issue will feature FORTH, and I hope to survey the PET FORTHs then.

Now let's take a look at why there is a need for all these other languages, what kinds of improvements they make, and what sacrifices are necessary.

What's Wrong with BASIC?

BASIC is inadequate in several areas:

1. It is too slow for many applications. Therefore, programmers must resort to writing machine language, which can be very difficult and time consuming.

2. It occupies too much memory, reducing the size of programs that can be executed.

3. It is often difficult to understand a BASIC listing. The necessary comments consume memory and slow execution speed, so they are often omitted.

4. It is not a "structured" language. Such features as global *vs.* local variables, named procedures, long variable names, and a logical program flow are quite foreign to most BASICs. Structured programs take longer to write, but the results pay off in a number of ways.

5. Although BASIC is a "universal language," the implementations of it are different. In other words, you can't just type into your PET a program that was written for another computer, without knowing a lot about that other BASIC dialect.

6. BASIC encourages sloppy programming with its convenience and lack of structure.

Of course there's a lot right with BASIC. The biggest advantage is that it comes with most microcomputers. Communication with peripheral devices and screen editing are usually much more difficult in the alternate languages. Everyone knows some BASIC, even if it's a slightly different dialect. BASIC is also easy to learn, and BASIC programs are easy to debug. The fact that it's so easy is reason enough to apply it in most situations. So if you aren't bothered significantly by any of the above "BASIC problems," by all means stay with BASIC.

Several of the alternate languages improve speed, consume less memory, and allow convenient manipulation of memory contents, but usually are more difficult to write and read. VIGIL, reviewed here in August, is such a language, oriented toward easy manipulation of graphics in game applications. FORTH is another, employing a threaded structure and a user stack. I'll begin my coverage of alternate languages with RPL, a new language designed to compete with FORTH.

RPL — from Samurai Software

RPL stands for Reverse Polish Language, as some of you may have suspected. It refers to the sort of backward notation used not only in this language, but also on Hewlett-Packard calculators and in FORTH. The key feature to all of these is the

(Continued on page 104)

FOCUS ON THE 6809 MICRO WORTH SWITCHING FOR

FOCUS ... ON ITS SUPERIOR HARDWARE

- 6809E Microprocessor provides 8-bit economy with 16-bit capabilities, position independent code and many advanced features
- Commercial quality Keyboard with full UPPER/lower case ASCII, numeric keypad and cursor control keys
- Two double sided, double density mini diskettes with 840K bytes of IBM compatible storage
- Full Video with programmable screen formats, bit mapped graphics, user definable character sets, reverse video and hardware scrolling, plus a light pen interface
- Full Communications support for RS-232 at 50 to 19.2K baud with programmable data formats
- Built-In I/O Drivers include 6 parallel I/O ports, 3 serial I/O ports, 6 timers/counters, 20 mA current loop, and a programmable cassette interface

FOCUS ... ON ITS EXTRAORDINARY SOFTWARE

- Choice of two 'Universal' 6809 Disk Operating Systems supported by many companies: FLEX ™ or OS-9™
- Extended Floating Point Disk BASIC Interpreted and Compiled
- Text Editor and Macro Assembler
- Full feature MicroMon debugging monitor
- Built-in Word Processor

FOCUS ... ON ITS VERSATILE EXPANDABILITY

- IEEE-488 Bus Controller option for instrumentation
- Supports multiple 8" diskette drives
- PASCAL, FORTH and other high level programming languages are available
- Complete Application Packages are available from many software sources
- Add-on Hardware includes: Date/Time, Opto-Isolators, Stepper-Motor Controller, A/D and D/A, RAM and EPROM Memory, and others from several manufacturers

FOCUS ... ON YOUR APPLICATION REQUIREMENTS

- FOCUS provides a completely integrated system of hardware and software, so that you can concentrate on your application requirements

FLEX is a trademark of Technical Systems Consultants
OS-9 is a trademark of Microware Systems Corporation
FOCUS is a trademark of The Computerist, Inc.

THIS IS FOCUS™

$3495.00

Very Complete

THE COMPUTERIST®

34 Chelmsford Street
Chelmsford, MA 01824
Phone: 617/256-3649 Telex: 955318 INTL DIV

MICRO

Microbes and Updates

Dear MICRO:

The utility program "Binary File Parameter List," by Clyde R. Camp (MICRO 38:45) produces an incorrect count of free sectors when used with DOS 3.3. The reason is the final statement in line 1500: V = INT (V/2). This drops the least significant bit in the two-byte map for each track, so that sector 0 is never counted. I have made a small change in line 1500 and added line 1505 to correct this (see below).

```
1500 FOR I = 56 TO 195
     STEP 4: S = PEEK (BASE
     + I) * 256 + PEEK (BASE
     + I + 1):V = INT (S / 2)
1505 IF S / 2 < > NT (S / 2)
     THEN CNT = CNT + 1
```

J. Morris Prosser
3157 Indian Village Rd.
Pebble Beach, CA 93953

```
]LIST1500,1505

1500  FOR I = 56 TO 195 STEP 4:S =
 PEEK (BASE + I) * 256 + PEEK
(BASE + I + 1):V = INT (S /
2)
1505  IF S / 2 < > INT (S / 2) THEN
CNT = CNT + 1
```

Dear MICRO:

I received the October issue of MICRO today containing my article "Solar System Simulation, Part 2," (41:108). I found one error in a DATA statement. Three values were duplicated. Below is the correct version of line 3330.

```
15,57,-22,-29,16,3,
-19,-40,16,18,-25,
-28,16,28,-26,-19,16,
33,-28,-7,16,47,-34,
-12,16,48,-37,-58,16,
50,-42,-17
```

Dave Partyka
1707 N. Nantuckett Dr.
Lorain, OH 44053

Dear MICRO:

I have just been re-reading my article "Interfacing Two 12-Bit A/D Converters to an AIM" (41:100) and I have noticed an error. In listing 2, the BASIC line 30 has been changed from X = USR(N), in my original text, to M = USR(N). This does not make any particular difference to operation, but in the text on page 105, second paragraph under BASIC Program, the sentence "The assignment of a value to X here..." would need to read, "The assignment of a value to M here..." to correspond.

G. Roger Heal
University of Salford
Salford M5 4WT,
Lancashire, England

Dear MICRO:

The program listing in "Sorting with Applesoft," by Norman P. Herzberg (MICRO 39:92) contains several errors. The corrected lines are:

```
750 TEMP = R(J):R(S(J)) =
    TEMP:R(J) = J:S(TEMP) =
    S(J):S(J) = J
2000 REM SORT
5050 DATA 169,76,141,245,3,
     169,58,141,246,3,169,3,
     141,247,3,96,32,227,223,
     133,133,132,134,32,190,
     222,32,227,223,160,2,
     177,133,72,177,131,145,
     133,104,145,131,136,16,
     243,96,0
```

J.C. Shellenbarger
1181 S. Sunkist St., Apt. 20
Anaheim, CA 92806

Dear MICRO:

In regard to the September issue, Clement Osborne's "Shaper" is fantastic. I tried several others and even wrote my own, but this beats all.

Here are a couple of additives which helped me through it, but they're not necessary for operation.

David L. Angell
18 Fairview Ave.
Cranston, RI 02905

```
]LIST 1030

1030  GOSUB 1325:N = N + OE

]LIST 1180

1180  TEXT : POKE 34,5:S = OE: GOTO 1050

]LIST 1325,1330

1325  VTAB 1: HTAB 1: PRINT "0-MOVE UP        4-PLOT & MOVE UP        1-MOV
     E RIGHT      5-PLOT & MOVE RIGHT     2-MOVE DOWN       6-PLOT & MOVE DOW
     N      3-MOVE LEFT       7-PLOT & MOVE LEFT"
1330  POKE 34,5: VTAB 24: RETURN

]LIST 6030,6032

6030  PRINT : PRINT "WHAT IS THE STARTING LOCATION OF TABLE ": PRINT : PRINT
     "(IF 0 THEN DECIMAL LOCATION IS 24567.)": PRINT : INPUT "       IN DEC
     IMAL -> ";SL
6032  PRINT : PRINT "DOUBLE CHECK STARTING LOCATION ! ": PRINT : INPUT "IS
      IT CORRECT ? ";A$: IF  LEFT$ (A$,OE) < > "Y" GOTO 6030
```

Dear MICRO:

I was very pleased to see my article "Monobyte Checksum Dumper" printed in the July issue of MICRO (38:67).

A few remarks, or corrections:

Listing 1: 1E2C C904 CMP #$04 should read 1E2C E004 CPX #$04.

The missing part of listing 1 was already corrected in Microbes and Updates (40:93). Here are the lines omitted from listing 2:

Peter D. H. Broers
Overijsselstraat 9
5144 EH Waalwijk, Netherlands

Broers' Listing 2

```
1F69 18         ADCHCK CLC            ;ADD THE BYTE TO THE CHECKSUM
1F6A 65E6              ADC CHCK
1F6C 85E6              STA CHCK
1F6E 9002              BCC *+4
1F70 E6E7              INC CHCK+1
1F72 60                RTS
1F73                   ;
1F73 208A1F     ADRIN  JSR DIGIN      ;GET 2 HEX DIGITS
1F76 0A                ASL            ;AND CALCULATE BYTE, STORING IT
1F77 0A                ASL            ;IN LOCATION "ADRES+Y"
1F78 0A                ASL
1F79 0A                ASL
1F7A 99E000            STA ADRES,Y
1F7D 208A1F            JSR DIGIN
1F80 19E000            ORA ADRES,Y
1F83 99E000            STA ADRES,Y
1F86 88                DEY
1F87 10EA              BPL ADRIN      ;REDO FOR Y+1 BYTES
1F89 60                RTS
1F8A                   ;
1F8A 20EBFF     DIGIN  JSR BYTIN      ;GET ONE HEX DIGIT
1F8D 20EEFF            JSR BYTOUT     ;DISPLAY IT
1F90 2093FE            JSR $FE93      ;TEST IT FOR VALID HEX AND MAKE BINARY
1F93 30F5              BMI DIGIN      ;0-15. IF NOT VALID, REDO.
1F95 60                RTS
1F96                   ;
1F96 A0FF       PRMPTS LDY #$FF       ;MESSAGE PRINTER "PROMPTS"
1F98 C8         PLOOPA INY            ;FIND MESSAGE NR. X
1F99 B9AE1F            LDA MESSAG,Y
1F9C D0FA              BNE PLOOPA
1F9E CA                DEX
1F9F D0F7              BNE PLOOPA
1FA1 C8         PLOOPB INY            ;AND PRINT (& SAVE?)
1FA2 B9AE1F            LDA MESSAG,Y
1FA5 F006              BEQ RETURN
1FA7 20EEFF            JSR BYTOUT
1FAA 4CA11F            JMP PLOOPB
1FAD 60         RETURN RTS
1FAE                   ;
1FAE 00         MESSAG BYT 00         ;MESSAGE 0
1FAF                   ;
1FAF 455252     MESSA  ASC 'ERROR << HIT G' ;ERROR MESSAGE
1FB2 4F5220
1FB5 3C3C20
1FB8 484954
1FBB 2047
1FBD 00                BYT 00         ;DURING THE LOADING
1FBE                   ;
1FBE 0A0D       MESSB  HEX 0A0D       ;MESSAGE 2--MESSAGE WHEN
1FC0 44554D            ASC 'DUMP B/M' ;STARTING THE DUMPER
1FC3 502042
1FC6 2F4D
1FC8 00                BYT 00
1FC9                   ;
1FC9 0A0D       MESSC  HEX 0A0D       ;MESSAGE 3--ASKING FOR
1FCB 465253            ASC 'FRST/LAST/AUTO?' ;THE ADDRESSSES
1FCE 542F4C
1FD1 415354
1FD4 2F4155
1FD7 544F3F
1FDA 0A0D              HEX 0A0D
1FDC 00                BYT 00
1FDD                   ;
1FDD 0A0D       MESSD  HEX 0A0D       ;MESSAGE 4--ASKING FOR A "Y"
1FDF 524541            ASC 'READY ?'  ;WHEN READY TO DUMP
1FE2 445920
1FE5 3F
1FE6 00                BYT 00
1FE7 0A0D       MESSE  HEX 0A0D       ;MESSAGE5--LOADER START
1FE9 062E31            STR '.1F00/'
1FEC 463030
1FEF 2F
1FF0 00                BYT 00
1FF1                   ;
1FF1 2E3146     MESSF  ASC '.1F00G'   ;MESSAGE 6--LOADER AUTOSTART
1FF4 303047
1FF7 00                BYT 00
1FF8                   ;
1FF8 0D         MESSG  HEX 0D         ;MESSAGE 7--BASIC POINTER
1FF9 2E3030            ASC '.0079/'
1FFC 37392F
1FFF 00                BYT 00
2000                   ;
                       END
```

A TEAM OF 6809 SUPERSTARS:
Smoke Signal's Chieftain™ Computer, and Software by Microware®

HERE'S THE *TOTAL* 6809-BASED SYSTEM FOR THOSE WHO DEMAND UNSURPASSED POWER, FLEXIBILITY AND RELIABILITY

After years of worldwide use in diverse and challenging applications, the outstanding performers in 6809 computer operations are SMOKE SIGNAL and MICROWARE. These leading companies are recognized as the undisputed choices **when there is no room for compromises**.

WHY SMOKE SIGNAL AND MICROWARE LEAD THE 6809 FIELD

Smoke Signal began pioneering research and development on 6800/6809-based computer systems back in 1977. Microware worked three years to perfect OS-9™ and BASIC09.™

Both companies have evolved outstanding 6809-based products from early engineering research, **and both pay almost fanatical attention to detail.** For example . . .

SMOKE SIGNAL'S 6809-based Chieftain™ computer series has **proven** its superiority in hundreds of demanding tasks. From gold-plated connectors to highest-quality materials throughout, each Chieftain™ is built to deliver absolute dependability from day one, and **stay** that way through years of service.

• ENDURANCE-CERTIFIED
. . . an exclusive Smoke Signal quality-control measure that positively verifies a component is free of defects, and meets or exceeds all specifications.

Every Chieftain™ is meticulously ENDURANCE-CERTIFIED at 2.2 MHz. That's SMOKE SIGNAL's endorsement of product perfection.

MICROWARE's state-of-the-art OS-9 UNIX*-like operating system and the BASIC09 language have been developed in close coordination with computer manufacturers to maximize optimum system performance. The finest possible support and documentation further ensure satisfaction. Microware software performance is best summed up in this remark by a 25-year computer veteran:

"BASIC09 IS THE FINEST HIGH-LEVEL LANGUAGE I'VE EVER SEEN IN THE INDUSTRY!"

Thousands of engineers and programmers use MICROWARE software products as their standard time-saving tool . . . to execute process-control applications . . . and for other vital functions. COBOL and PASCAL are also available under the OS-9 operating system.

HOW THIS REMARKABLE TEAM OF COMPUTER SUPERSTARS CAN SERVE *YOU*

SMOKE SIGNAL's Chieftain™ computer provides an array of configurations ranging from 5¼-inch drives for single-user applications to multi-user, multi-tasking capabilities. Winchester hard-disk drive systems are also available.

In other words, **breathtaking power** with as little as 48k memory; Microware's OS-9 Level Two can access up to one full megabyte that your Chieftain™ can address!

One more sampling of the awesome processing potential at your fingertips with the Smoke Signal Chieftain™ computer:

MICROWARE'S Stylograph screen-oriented word processing package instantly makes Chieftain™ an easy-to-use document preparation system with comprehensive editing commands.

THERE'S MUCH, MUCH MORE! Call or write SMOKE SIGNAL for details on Chieftain™ computers and MICROWARE software.

SMOKE SIGNAL Dealer opportunities are still available . . . please request information.

*UNIX is a trademark of Bell Telephone Laboratories.

SMOKE SIGNAL BROADCASTING

MICROWARE
P.O. Box 4865
Des Moines, Iowa 50304

31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

☐ Send information about Chieftain™ computers and Microware software.

☐ Provide information about Smoke Signal's Dealer program.

Name ____________________

Address ____________________

City ____________ State ________ Zip ________

Telephone ( ) ____________________

From Here to Atari

James Capparell
297 Missouri
San Francisco, California 94107

Last month I showed you how to use the Load Memory Scan (LMS) instruction of the display list to effect a scrolling screen. Recall that the display list is the set of instructions used to control an LSI chip called ANTIC. ANTIC, a dumb microprocessor, functions as a graphics controller. Its principle functions are to specify the location in memory to he displayed, the mode of display (14 graphics/text modes witb differing resolutions to choose from), horizontal/vertical scroll enable (discussed last month) and display list instruction interrupt enahle.

This month I've included an ANTIC disassembler. This program requires you to enter a BASIC graphics mode numbered 0 - 8, and will then locate the associated display list and decode the instructions. Note that this program prints the ANTIC display modes numbered 2 - 15. Use the program and the ANTIC/BASIC correspondences will hecome apparent. (See program 1.) I also want to take you on a short trip into the world of basic raster scan graphics, Atari style, and then provide a quick lesson in the use of display list interrupts.

The normal NTSC raster television is made up of 625 interlaced scan lines. These scan lines are tbe horizontal lines appearing in the picture tuhe phospher when energized by the electron beam as it sweeps left to rigbt, top to bottom, across your screen. Interlacing occurs in normal television to eliminate flicker. It simply means that all even scan line rows are "painted" in one frame, and all odd lines in the next. The frame refresh rate is 60 Hz.

Each Atari frame image contains 262 scan lines with no interlacing. Every frame is the duplicate of the prior one unless there is programmer intervention. The image is repainted 60 times per second, and the electron beam is turned off at the end of every scan line. At that time it is returned to the left edge of the screen to start the next line trace. This is called horizontal blank time.

The beam is also turned off after every frame so that it may return to top left corner of the screen, called vertical blank time. These two time periods are very important to the would-be animator. It is crucial to understand how much time is available and how to enter code such that it will be executed at the appropriate moment.

The 6502 microchip in the Atari cycles at 1.79 megahertz, almost twice as fast as the normal 6502. This cycle rate was chosen so that two color clock widths on a scan line equal one machine cycle. There are 228 color clocks on every scan line, and the maximum displayahle width of any scan line is 176 color clocks, called "wide playfield" in the Atari literature. The maximum resolution is ½ color clock, and therefore Atari can display up to 352 picture elements (pixels) horizontally. The maximum vertical resolution, in scan line units is 240. Effectively, Atari has a high-resolution mode of 352 × 240.

It's important to realize that there are physical limitations to this size display. Depending on your television's adjustment, some of the displayed image may appear on the curved edge of the picture tuhe. This overlap is called overscan. While overscan is not important in normal television viewing, it is crucial when your word processor is printing what you can't see.

Atari, in its Operating System (O.S.), used a more conservative screen size of 320 (160 clocks) horizontally by 192 scan lines vertically. This width screen is called normal playfield in the documentation. In this way Atari defeated normal overscan and assured us of seeing an entire image. There is a narrow playfield width as well, 256 pixels (128 clocks wide). These dimensions and timing are important since wbat is not used at display time is left over and available at interrupt time. (See table 1 for timing.)

It is relatively simple to change between screen widths. Location $22F

(Continued on page 46)

***Table 1:* Timing**

1.79 MHZ machine cycle
262 scan lines per frame
228 color clocks per scan line
60 frames per second refresh rate

1.79/60 = 29868 machine cycles per frame
29868/262 = 114 machine cycles per scan line
228/114 = 2 color clocks per machine cycle

Vertical Blank Time

262 scan lines - 192 displayed scan line = 70
70 × 114 cycles/line = 7980 cycles available*

Horizontal Blank Time

Wide Playfield
228 clocks - 192 clocks = 36 clocks
36/2 = 18 machine cycles

Normal Playfield
228 clocks - 160 clocks = 68 clocks
68/2 = 34 machine cycles

Narrow Playfield
228 clocks - 128 clocks = 100 clocks
100/2 = 50 machine cycles

*All graphics are cycle-stealing Direct Memory Access (DMA). Depending on graphics mode and memory refresh, this value will he less.

(Continued from page 43)
controls playfield width. Called SDMCTL in the documentation, it is initialized to $22. Writing a $23 will change the screen dimension to wide, and writing $21 will reduce the screen to narrow. SDMCTL is the O.S. shadow for a hardware register in the ANTIC chip at $D400, called DMACTL.

Since many of these hardware locations are write only, the O.S. keeps copies, called shadows, in RAM. Shadow registers update the associated hardware at Vertical Blank Interrupt time. Remember to use the shadows to effect a permanent change to the entire frame. The exception occurs when using a display list interrupt. These interrupts can occur, under programmer control, on any scan line of every frame. To effect an immediate change at scan line interrupt, you must write directly to the hardware register.

To use the Display List Interrupt (DLI), a number of things must be accomplished. First, write the DLI service routine. The important thing to

```
10 REM *** PROG1 ***
20 REM MEMORY AND DISPLAY LIST VARIES WITB GRAPHICS MODE
30 REM DUMP AND DISASSEMBLE DISPLAY LIST
40 REM
100 ? " INPUT GRAPHICS MODE ";:INPUT MODE
105 GRAPHICS MODE
110 LST=PEEK(560)+PEEK(561)*256:REM FIND START OF DISPLAY LIST
120 MEMRY=PEEK(LST+4)+PEEK(LST+5)*256:REM FIND START OF DISPLAY MEM.
130 RAMTOP=PEEK(106)*256:REM NUMBER OF PAGES IN MEM DEFINED AT POWER ON
140 REM LIST
150 LPRINT " OS GRAPHICS MODE ";MODE
160 LPRINT " RAM AVAILABLE AT POWER ON ";RAMTOP
170 LPRINT " START OF DISPLAY LIST ";LST
180 LPRINT " START OF DISPLAY MEMORY ";MEMRY
190 REM DUMP DISPLAY LIST WITH DISASSEMBLY OF INSTRUCTIONS
195 LMS=64:INT=128:HSCRL=16:VSCRL=32:JVB=65:JMP=1
200 FOR I=LST TO MEMRY-1
205 LPRINT I;"  ";PEEK(I);
210 INST=PEEK(I):REM DISPLAY LIST VALUE
215 IF INST>=128 THEN GOSUB 1100:GOTO 400
220 GOSUB 1140
400 NEXT I
410 STOP
1100 INST=INST-INT:REM GET RID OF INTERRUPT BIT
1105 LPRINT " INSTRUCTION INTERRUPT ENABLE "
1140 GOSUB 2000:REM FIND JUMPS AND BLANKS
1150 IF INST=0 THEN RETURN
1160 GOSUB 1400:REM GO FIND LMS
1170 GOSUB 1500:REM GO FIND VSCROL
1180 GOSUB 1600:REM GO FIND HORIZONTAL SCROLL
1190 GOSUB 1700:REM TRANSLATE ANTIC MODE TO OS GRAPHICS MODE
1200 RETURN
1400 IF INST<66 THEN RETURN :REM NO LMS
1405 LPRINT " LOAD MEM SCAN FROM ";PEEK(I+1)+PEEK(I+2)*256
1410 INST=INST-LMS:REM GET RID OF LMS BIT.
1420 I=I+2:REM INCREMENT LOOP AROUND ADDRESS BYTES
1430 RETURN
1500 IF INST<34 THEN RETURN :REM NO VSCROL ENABLE
1510 INST=INST-VSCRL:REM GET RID OF VSCROLL BIT
1520 LPRINT " VERTICAL SCROLL ENABLED "
1530 RETURN
1600 IF INST<18 THEN RETURN :REM NO HSCROLL ENABLE
1610 INST=INST-HSCRL:REM GET RID OF HORIZONTAL SCROLL BIT
1620 LPRINT " HORIZONTAL SCROLL ENABLED "
1630 RETURN
1700 LPRINT " ANTIC DISPLAY MODE ";INST
1750 RETURN
2000 IF INST=O OR INST=16 OR INST=32 OR INST=48 OR INST=64 OR INST=80 OR INST=96 OR INST=112 THEN G
OSUB 2100
2010 IF INST=1 THEN GOSUB 2200
2020 IF INST=65 THEN GOSUB 2300
2030 RETURN
2100 LPRINT " BLANK ";INT(INST/16)+1;" LINES"
2110 INST=0:RETURN
2120 REM
2200 LPRINT " JUMP INSTRUCTION TO ";PEEK(I+1)+PEEK(I+2)*256
2210 I=I+2:REM INCREMENT AROUND ADDRESS BYTES
2215 INST=INST-JMP:RETURN
2220 REM
2300 LPRINT " JUMP & WAIT FOR VERTICAL BLANK TO ";PEEK(I+1)+PEEK(I+2)*256
2310 I=I+2:REM INCREMENT AROUND ADDRESS BYTES
2315 INST=INST-JVB:RETURN
```

remember here is to save and restore any registers needed by the routine. Then find a free place in memory for this routine. (As you know, Atari has reserved page six, decimal 1536, just for users.) Next, update the vector at $200 and $201 to point to start of the routine. Now change the appropriate display list instruction to cause an interrupt (accomplished by turning on bit 7 of the instruction). Finally, enable DLIs by setting bit seven of hardware register $D40E, called NMIEN (Non-Maskable Interrupt enable). See program 2 for a simple example.

Also remember to set the interrupt in the mode line prior to the location where you would have the changes occur. Then write to a location called WSYNC $D40A. This will cause any changes to be delayed to the start of the next scan line and, therefore, allow a smooth synchronized transition.

DLIs can be used for everything from putting many colors on the screen, to changing among a number of character sets, to moving player/missiles around. To get the most from Atari, experiment with this concept.

```
10 REM *** PROGRAM 2 ***
20 REM THIS WILL CREATE A DISPLAY LIST WITH DLI ENABLED
30 REM THE SCREEN WIDTH IS NARROWED AT DLI TIME AS WELL
40 REM
45 GRAPHICS 0:SETCOLOR 4,4,9:REM SET BORDER COLOR
50 DLST=PEEK(560)+PEEK(561)*256:REM FIND START OF DISPLAY LIST
60 POKE DLST+14,PEEK(DLST+14)+128:REM TURN ON INTERRUPT BIT 7
70 FOR L=0 TO 29:REM POKE DLI SERVICE ROUTINE INTO PAGE 6
80 READ INSTRCT:POKE 1536+L,INSTRCT
90 NEXT L
100 DATA 72,138,72,169,40,162,48,141,10,212,141,23,208
110 DATA 142,24,208,169,33,141,0,212,162,140,142,26,208,104,170,104,64
120 POKE 512,0:POKE 513,6:REM POINT TO DLI INTERRUPT SERVICE ROUTINE
130 POKE 54286,192:REM ENABLE DLI
140 LIST
150 REM *** DLI SERVICE ROUTINE ***
152 REM PHA       SAVE REGISTERS
154 REM TXA
156 REM PHA
158 REM LDA #$28  CHARACTER LUMINENCE
160 REM LDX #$30  BACKGROUND COLOR
162 REM STA $D40A WAIT FOR HORIZONTAL SYNCH
164 REM STA $D017 PLAYFIELD 1
166 REM STX $DO18 PLAYFIELD 2
168 REM LDA #21   NARROW PLAYFIELD
170 REM STA $D400 DMACTL ENABLE NARROW WIDTH
172 REM LDX #$8C  BORDER COLOR
174 REM STX $D01A COLBK
176 REM PLA       RESTORE REGISTERS
178 REM TAX
180 REM PLA
182 REM RTI       RETURN FROM INTERRUPT
```

MICRO

Introducing the M line . . .

Now! Drive Systems for AIM, KIM and SYM Computers — from PERCOM.

Assembled and tested systems start at only $599.95, including the drive controller circuit card, disk-operating system, interconnecting cable, drive and comprehensive users manual.

- **The right storage capacity** — Available in 1-, 2- and 3-drive systems, with either 40- or 80-track drives.
- **Flippy storage** — Flippy drives (optional) let you flip a diskette and store data and programs on the second recording surface.
- **High Storage Capacity** — Formatted, one-side storage capacity is 102 Kbytes (40-track drive), 205 Kbytes (80-track drive).
- **Proven Controller** — The drive controller design is the same as the design used in the Percom 680X LFD mini-disk system. This system — introduced in 1977 — has given reliable service in thousands of applications. Two versions are available: the MFD-C65 for the AIM-65 expansion bus, and the MFD-C50 for the System-50 (SS-50) bus.
  - Includes an explicit data separator circuit that's reliable even at the highest bit densities.
  - Provides for on-card firmware.
  - Includes a motor inactivity time-out circuit.
  - Capable of handling up to four drives.
  - Capable of reading both hard- and soft-sectored diskettes.
- **DOS included** — The MFD disk-operating system works with the AIM monitor, editor, assembler, Basic and PL/65 programs, interface is direct, through user I/O and F1, F2 keys. Diskette includes DOS source code and library of 20 utility commands.
- **Reliability assurance** — Drives are burned-in 48 hours, under operating conditions, to flag and remove any units with latent defects.
- **Full documentation** — Comprehensive hardware and software manuals are included with each system.

Now! Expand your AIM-65 with Low-cost System-50 Modules.

The Percom M65/50 Interface Adapter connects your M-65 bus to Percom's System-50 (SS-50) motherboard, allowing you to expand your AIM, KIM or SYM with proven System-50 modules. You can add disk storage, memory modules, even a video display system. The M65/50 provides buffer-amplification of address, data and control lines. On-card decode circuitry lets you allocate address space either to the computer or to the expansion motherboard. Price: only $89.95, *including* System-50 motherboard.

System Requirements: AIM-65, KIM or SYM computer with expansion bus and four Kbytes RAM (min).

Yes . . . I'd like to know more about Percom MFD drive systems. Rush me free literature.

Send to
PERCOM DATA COMPANY, Inc., Dept. 65-M
11220 Pagemill Rd. Dallas, TX 75243

name

address

city state

zip phone number

MAIL TODAY!

PERCOM

PERCOM DATA COMPANY, INC.
11220 PAGEMILL RD. DALLAS, TX 75243
(214) 340-7081

Toll-Free Order Number: **1-800-527-1222**

PRICES AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE.

Some Help for KIM

Part 2

Lest month we sew how the KIM memory dump routine could be improved to provide e progrem formet memory dump routine. This month we will continue our improvements to KIM by investigeting the operetion of the Single Step feeture. Next month we will investigete improving the feetures of the Single Step routines.

Wayne D. Smith
Math/Computer Science Dept.
Austin Peay State University
Clarksville, Tennessee 37040

The single-step mode on the KIM can be very useful in determining where an erroneous program is malfunctioning. When KIM is in the single-step mode, one instruction is executed eacb time the letter G is depressed on the terminal. After the step is executed, KIM prints the address of the next instruction to be executed, and the operation code at that address. KIM then awaits another letter G key-press before executing the next step.

This mode of operation can be a great help to the user, but unfortunately, it does not always provide enough information for complete analysis of program operation. For example, the operand associated with the instruction ahout to be executed isn't shown. A user can single-step through a program several times hefore he discovers that all the operation codes are correct, but that one of tbe addresses is wrong.

Even the operation code and the operand are often not sufficient to pinpoint the error. It would also be heneficial to he able to determine the contents of the registers, the stack pointer and the status flags. This information is available to the user, hut he must first remember when KIM stores this information, and then print these locations one at a time. It would he much more convenient if the single-step software would print all this information for the user after each step were executed.

To write a single-step program, however, it is first necessary to understand how the KIM single-step feature operates. In essence, whenever an instruction is fetched and the single-step switch is on, a non-maskable interrupt is generated. This interrupt is generated hy the sync signal from the 6502, which goes high only when an instruction fetch is taking place. This signal remains low for all other memory access operations.

The non-maskable interrupt input to the 6502 is an edge-triggered signal. That is, this pin is sensitive only to a high to low transition of the input signal. This means that the interrupt is not generated until the sync signal goes high and then low again. When this signal goes low, however, the instruction fetch is already in progress. Therefore, the interrupt is not honored until the current instruction has been fetched and executed. After instruction execution has heen completed, the interrupt is honored, and the normal interrupt sequence is entered. The program counter and status register are pushed onto the stack. An indirect jump is then executed to the address stored in the NMI vector address ($17FA and $17FB on an unexpanded KIM). Normally, this is a jump to location 1C00.

The software located at 1C00 takes care of storing the registers, the program counter, the stack pointer and the status register in predetermined page zero addresses. After printing a carriage return and a line feed, KIM then prints the address of the next instruction and the value stored at that address. This is relatively easy, since the program counter that was stored earlier is pointing to that address. KIM then goes into a loop awaiting a new key-press.

But, the question arises as to how the software located at 1C00 should he executed without generating additional interrupts whenever an instruction fetch takes place. This prohlem is eliminated by NANDing the sync signal with the K7 signal and using this output to generate the interrupt. In this manner, an interrupt will only take place if an instruction fetch takes place to an address which is outside K7. If the K7 signal is low, the interrupt gate is effectively disabled. This means that any of the KIM software located in the ROM address space K7 may be executed without generating interrupts. This is very convenient. Not only is the single-step software located in K7, hut most of the other KIM routines, except the tape input and output programs, are also located here. This point does raise a minor problem because it is not possible to single-step any program which is located in K7. This presents no real difficulty, however, since I have yet to find a programming error in the KIM software.

Now, if the user wishes to write an improved single-step routine, it will not work if it is located outside the K7 address space. In fact, if this is attempted, an infinite loop of interrupts will be generated as the interrupt causes a hranch to a location, which generates an interrupt which causes a branch to ..., etc.

Therefore, to use a modified single-step program, there are two possihle alternatives. The first alternative is to locate the new program in K7. This is clearly impossihle, since K7 is not only ROM, but all 1024 locations are already in use by the KIM software. The only other alternative is to make a minor hardware modification to the KIM itself so that programs in some other area will not generate the NMI signal when instructions are fetched from this area.

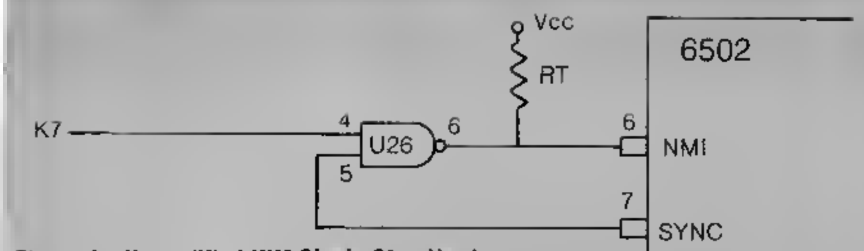


Figure 1e: Unmodified KIM Single Step Herdware

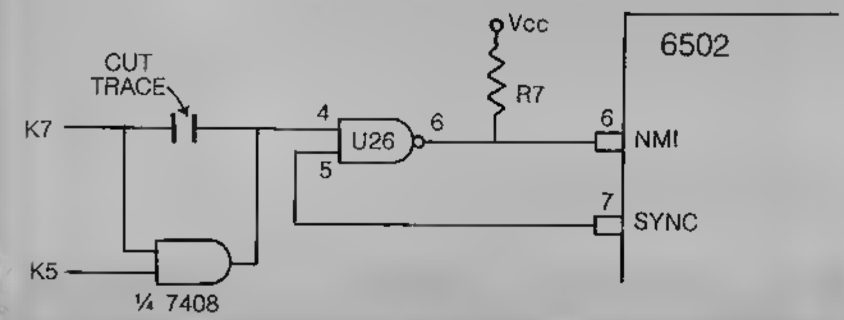


Figure 1b: Modified Hardware for Single Step Program Located In K5

Figure 1a shows tbe hardware associated witb the KIM single-step interrupt generator. To prevent generating an interrupt in an area other than K7, it is necessary to AND the K7 signal with the K signal for that area of memory. Notice that it would not be wise to replace the K7 signal with the new signal, since this would make all the KIM routines in K7 run in single-step mode. Since tbe terminal I/O routines would probably be needed for any single-step program, this approacb is impractical.

Figure 1b shows the added gate which allows location of tbe single-step program in either K7 or K5. In my case, K5 was chosen because the 128 bytes of RAM located in this area are just enough for the single-step program, but too small for anything else. In addition, I also use area K5 for all my I/O ports, and hence, have no additional RAM located here. Access to an I/O port will not generate a fetch (sync) signal, therefore K5 is an ideal area for my system.

If you prefer to locate the single-step program somewhere else, you may do so by substituting the appropriate K signal for K5. Tbis will definitely become necessary if you are using a TVT-6, which utilizes the 128 bytes of K5 RAM to generate video displays. The single-step program itself is easily relocated, but remember, if you move it to another area, no program which is stored in tbat entire 1K area can be single-stepped.

Next month we will look at some variations on the hardware modifications, and also examine the software needed to provide an improved single-step capability.

MICRO

OSI TRS-80 COLOR-80 OSI

GALAXfAN · 4K · One ol the fastest and finest arcade gemes ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For thosa who loved (and tired of) Alien Invadars. Specify system — A bargain at $9.95 OSI

LABYRINTH · 8K · This has e display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monstars on foot. Checking out and testing this one was tha most fun I've had in yearsl — $13.95. OSf

THE AARDVARK JOURNAL

FOR OSf USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modily the system, and reviews of OSI related products. In the last two yaars we have run articles like thesel

1) A tutorial on Machine Coda for BASIC programmers.

2) Complete listings of two word processors for BASIC IN ROM machines.

3) Moving the Directory off track 12.

4) Listings for 20 game programs for tha OSI.

5) How to write high speed BASIC — and lots more —

Vol. f (1980) 6 back issues · $9.00

Vol. 2 (f98f) 4 back issues and subscription for 2 additional issues · $9.00.

ADVENTURES!!!

For OSI, **TRS-80**, and **COLOR-80**. These Adventures are written in BASIC, are full featured, last action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commends like "Look in tha Coffin" and "Light the torch".)

Adventures require 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 aach.

ESCAPE FROM MARS (by Rodger Olsen)

This ADVENTURE takes place on the RED PLANT. You'll have to axplore a Martian city and deel with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)

This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and toughl

TREK ADVENTURE (by Bob Retelle)

This ona takes placa aboard a familiar starship. The crew has left lor good reesons · but thay forgot to take you, and now you ara in deep trouble.

DEATH SHfP (by Rodger Olsen)

Our first and original ADVENTURE, this one takes place aboard a cruisa ship · but it ain't the Love Boat.

VAMPfRE CASTLE (by Mike Bassman)

This is a contest between you and old Drac · and it's getting a littia dark outside. $14.95 each.

OSI NEW-NEW-NEW OSf
TINY COMPILER

Tha aasy way to spaad In your programs. Tha tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs lrom 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It Is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, ·, =, *, /, , <, > , Varieble names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmanted by the user. It comes with a 20 page manual.

TINY COMPILER — $f9.95 on tape or disk OSI

SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing lor —

SUPERDISK II $29.95 (5¼") OSI

BARE BOARDS FOR OSI C1P

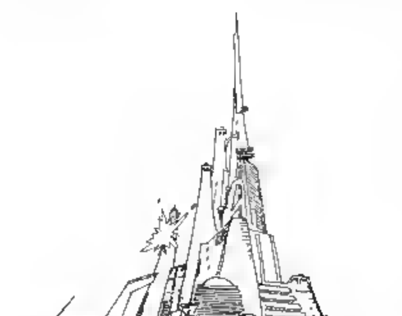
MEMORY BOARDS!!! — for tha C1P — and they contain parallel portsl

Aardvarks new memory board supports BK of 2f14's and has provision for a PIA to give a parallal ports! It salls as a bera board lor $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available nowl

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adept our C1P boards to your C4/8P. · $14.95.

16K RAM BOARD FOR C1P — This one does not hava a parallel port, but it does support 16K of 2f14's. Bare Board $39.95.



WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a lina-oriented word processor designad for the offica that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any taxt singla, double or tripla spacad and has text centering commands. It will make any number of multiple copies or chain files together to print an entlra disk of deta at one time.

MAXI-PROS has both global and line adit capability and the pollad keyboard varsions contain a corrected keyboard routine that maka the OSI keyboard dacode as a standerd typewriter keyboard.

MAXI-PROS also has sophisticated file capabibilitias. It can access a file for names and addressas, stop for Inputs, and print form letters. It hes fila merging capabilities so that it can stora and combine paragraphs and pages in any order.

Best of all, it is In BASIC (0S65D 51/4" or 8" disk) so that it can be aasily adaptad to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS — $39.95. Specify 5¼ or 8" disk.

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds lIne edit functions, software selectable scroll windows, bell support, choice of OSI or standard kayboard routinas, two callable screan clears, and softwara support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. CfP requires only a jumper changa. — $39.95

C1E/C2E similar to above but with extendad machine code monitor. — $59.95 OSI

ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)

TIMETREK · A REAL TIME, REAL GRAPHICS STARTRECK. Sea your torpedoes hit and watch your instruments work in real tima. No more unreelistic scrolling displaysl $14.95.

STARFfGHTER · This one man space war gama pits you ageinst spacecruisers, battlewagons, and one men fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Anothar real time goody. $9.95

BATTLEFLEET · This grown up version of Battleship is the toughest thinking gama available on OSI or 80 computers. Thare is no luck involved as you seak out the computers hidden fleet. A topographical toughie. $9.95

QUEST · A NEW IDEA IN ADVENTURE GAMES! Differant from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gathar men and supplies by combbat, bargaining, exploration of ruins and temples and outright banditry. When your lorce is strong enough, you attack tha Citadel of Moorlock in a life or death battle to the finish. Playabla In 2 to 5 hours, this one is different every time.

16K COLOR-80 OR TRS-80 ONLY. $14.95

Please specify system on all orders

This Is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSf systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.

AARDVARK TECHNICAL SERVICES, LTD.
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

OSI COLOR-80

OSI Symbolic Disassembler

This modification of Werner Kolbe's "Symbolic Disassembler" was written for OSI C4P. However, it should run, with few further modifications, on other OSI machines.

David E. Pitts
16011 Stonehaven Dr.
Houston, Texas 77059

I was in the process of trying to understand BASIC on my OSI 4PMF and had already ordered the books on how Microsoft BASIC works, but wanted to know more. I thought that a good way to learn would be to disassemble parts of BASIC (located $0200 to $2300), so I booted the system, loaded the assembler/disassembler and proceeded to disassemble that region of RAM. While the computer was churning away I glanced in the OSI 4P hook to be sure that I was working on the correct region. Much to my dismay I noticed that $200 to $2300 was now occupied by the Assembler and Extended Monitor. Back at the drawing board, three choices came to mind: move the Assembler/Disassembler, move BASIC, or write a disassembler in BASIC. The first two choices would involve considerable work in changing absolute addresses and jumps, so a disassembler in BASIC seemed in order.

My disassembler was just coming to life when my son pointed out the PET symbolic disassembler in the January 1981 issue of MICRO (32:23). Not being familiar with the dialect of BASIC used with the PET, I ignored his suggestion and continued on my program. A week later my disassembler was still giving OM (out of memory) errors after successfully disassembling about 6 lines of code — obviously due to a bug in the string usage, as there should have been plenty of the 48K of memory left. I was about ready to convert my string usage to the more efficient techniques discussed by Edward Carlson in "A 6502 Assembler in BASIC," MICRO, (34:7), when my son once again suggested the PET disassembler program.

The next few hours involved reading Werner Kolbe's carefully documented article, and keying in the program. Some confusion occurred when I encountered "GET" and the symbols for "home", but logic suggested the former was an input from the keyboard and the latter was a key on the keyboard, so I proceeded on that assumption. It was unclear why I would need to open and close files (line 45), as Kolbe did on the PET, unless I needed to save all the disassembled code. Since I didn't think that this was the case, I gambled that this was just a peculiar aspect of PET BASIC and removed the "OPEN", "CLOSE", and converted "PRINT#1" to "PRINT" in all the statements.

Editor's Note: Files were used in Kolbe's program to make it easy to switch between screen and printer. The PET screen is treated as an IEEE-488 device (3), just like the printer, so the save file number (1, in this case) can be used in the PRINT# statements. The only difference is in the device number when the file is opened.

Table 1 lists the line numbers and the changes that were made to convert to OSI BASIC. The converted program (shown in the listing) occupies 4067 bytes, which should allow sufficient room for table and string storage even on 8K machines. The program was written in DOS 3.2 OSI BASIC. The string bug problem on some BASIC-in-ROM machines would be the only

***Table 1:* Changes to the PET Disassembler**

Line	Remarks
10	remove the PET string bug fix
15	remove the PET string bug fix
40	add line for title
45	print code for output devices
50	change prompts
51	move "if statement" from line 50 to 51
105	change "GET" to PEEK(57088) = 5 for left shift
115	change to prompt for instructions
116	check for escape key, PEEK(57088) = 33
119	add to check for right shift, PEEK(57088) = 3
125	change from RETURN to GOTO116
165	change "=" to "#$"
295	change to E = PEEK(P)
550, 565, 575	remove SPC(8 – LEN(E$)); to cause variable and address list to be printed in 2 columns
all lines	change PRINT #1 to PRINT
471	additional lines had to be added
481	to data statements because of length
486	"
491	"
501	"
511	"
516	"
531	"
536	"

potential reason for it not working on all OSI 6502 systems.

I have changed the program to allow the "left shift" to stop the disassembly, "right shift" to resume, and "escape" to allow the user to list the addresses of the machine-generated labels. The program prompts the user for the output devices: two for the CRT and ten for the CRT and unit four (parallel) printer. Should you have a serial printer, three should be used in place of ten.

As in Kolhe's program, two passes through the program are necessary to get all the symbolic labels. I usually make the first pass using the CRT as the output device, and the second pass with the printer activated. Then I print out the symbolic labels and their addresses. The progam runs quite fast, disassembling eight pages of code (2K bytes) in about seven minutes. When the Centronics 737 printer is used, the system takes about 35 minutes for eight pages.

David Pitts is an Aerospace Technologist at the NASA-Johnson Space Center in Houston, Texas. His training is in Engineering Physics, Geophysics, and Meteorology, and he has been involved in the Gemini, Apollo, Skylab and Landsat spacecraft programs. He has programmed in FORTRAN on IBM, Univac, and DEC machines since the 1960's and only recently has been programming in BASIC.


```
2 REM SYMBOLIC DISASSEMBLER FOR PET BY WERNER KOLBE-MICRO,JAN81-PG 23
3 REM AS MODIFIED BY DAVID PITTS FOR OSI 4PMF
5 DM=255:PM=50
20 DIMM$(255),L$(DM),L(DM),Z%(255),Z$(255),P(PM),P$(PM)
25 DM=DM-1:PM=PM-1:Z$(0)="ZERO":L$(0)="LABEL":P$(0)="PAGE >0"
30 FORI=0TO255:READM$(I):NEXT
40 PRINTTAB(25);"DISASSEMBLER":PRINT:PRINT:PRINT
41 PRINT"<LSHIFT> TO STOP":PRINT
45 INPUT"OUTPUT DEVICE (2=CRT, 10=CRT & LINE PRINTER)";D:POKE8994,0
50 FL=1:INPUT"STARTING HEX LOC (OR LABEL CODE)";E$:PRINT
51 IFE$="PM"THENV$="":GOTO545
55 IFE$="PL"THENV$="L":GOTO545
60 IFE$="PJ"THENV$="J":GOTO545
65 IFE$="PZ"THEN560
70 IFE$="PW"THEN570
72 IFE$="ENTRY"THEN600
75 GOSUB280:P=E-1
80 P=P+1:E=P:S=1:GOSUB325:IFL$<>" "THENFL=1
85 GOSUB300
90 PRINTSPC(5-LEN(E$))E$;:GOSUB295:PRINT" "E$;:K=5
92 M$=LEFT$(M$(E),3):B=VAL(MID$(M$(E),4))
95 ONB*FL+1GOSUB135,165,170,180,185,190,210,215,220,235,240,250,260,265
105 IFPEEK(57088)=5THENGOTO115
110 GOTO80
115 PRINT:PRINT"<ESC> FOR LABEL ADDRESSES, <RSHIFT> TO CONT"
116 IFPEEK(57088)=33THENPRINT"LABEL CODE=PM,PL,PJ,PZ,PW":GOTO50
119 IFPEEK(57088)=3THEN105
125 GOTO116
130 PRINT" "E$SPC(K)L$SPC(7-LEN(L$))M$" ";:RETURN
135 IFFL=1ANDM$<>"?"THENPRINTSPC(8)L$SPC(7-LEN(L$))M$:RETURN
140 FL=0:IFM$="BRK"THENFL=1:PRINT:RETURN
145 PRINTSPC(15)"? : "CHR$(34);:IFE>30ANDE<128THENPRINTCHR$(E)
150 IFE<30THENPRINTCHR$(E+64)
155 IFE>127THENPRINTCHR$(E-128)
160 RETURN
165 GOSUB290:GOSUB130:PRINT"#$"E$:RETURN
170 V$=""
175 GOSUB290:GOSUB130:GOSUB380:PRINTZ$V$:RETURN
180 V$=",X":GOTO175
185 V$=",Y":GOTO175
190 V$=""
195 GOSUB290:PRINT" "E$;:H$=E$:GOSUB290:K=2
200 GOSUB130:E$=E$+H$:GOSUB280:IFP$<>""THENPRINTP$V$:RETURN
205 PRINTE$V$:RETURN
210 V$=",X":GOTO195
215 V$=",Y":GOTO195
220 GOSUB290:A1=E:GOSUB130:E=A1+P+1:IFA1>127THENE=P-255+A1
225 V$="L":S=0:GOSUB325:IFBTHENPRINTL$:RETURN
230 GOSUB300:PRINTE$:RETURN
235 V$=",X)":GOTO245
240 V$="),Y"
```

```
245 GDSUB290:GDSUB130:CDSUB3B0:PRINT"("Z$V$:RETURN
250 GDSUB290:PRINT" "E$;:H$=E$:COSUB290:K=2:COSUB130
255 PRINT"("E$H$")":RETURN
260 PRINTSPC(8)L$SPC(7-LEN(L$))M$"A":RETURN
265 CDSUB290:PRINT" "E$;:H$=E$:CDSUB290:K=2:CDSUB130:E$=E$+H$:GDSUB2B0
270 V$="J":S=0:CDSUB325:IFL$<>" "THENPRINTL$:RETURN
275 PRINTE$:RETURN
280 E=0:FDRI=1TDLEN(E$):B=ASC(MID$(E$,I,1))-48:IFB>9THENB=B-7
285 E=E*16+B:NEXT:RETURN
290 P=P+1
295 E=PEEK(P)
300 B=E:E$=""
305 H=INT(B/16):B=INT(B-16*H):B$=CHR$(B+48):IFB>9THENB$=CHR$(55+B)
310 E$=B$+E$:IFH>=1THENB=H:CDTD305
315 IFLEN(E$)<2THENE$="0"+E$
320 RETURN
325 B=-1:H=LL+1
330 I=INT((H+B)/2):IFL(I)=ETHENB=1:L$=L$(I):RETURN
335 IFL(I)>ETHENH=I:CDTD345
340 B=I
345 IFABS(N-B)>1THEN330
350 IFSOR(LL>DM)THENB=0:L$=" ":RETURN
355 LL=LL+1:IFL(I)<ETHENI=I+1
360 FDRB=LLTDI+1STEP-1:L(B)=L(B-1):L$(B)=L$(B-1):NEXT
365 L(I)=E:L$(I)=V$+MID$(STR$(LL),2)
370 B=0:L$=L$(I):IFE>PTHENB=1
375 RETURN
380 B=-1:H=ZZ+1
385 I=INT((H+B)/2):IFZ%(I)=ETHENZ$=Z$(I):RETURN
390 IFZ%(I)>ETHENH=I:CDTD400
395 B=I
400 IFABS(H-B)>1THEN385
405 ZZ=ZZ+1:IFZ%(I)<ETHENI=I+1
410 FDRB=ZZTDI+1STEP-1:Z%(B)=Z%(B-1):Z$(B)=Z$(B-1):NEXT:Z%(I)=E
412 IFS=2THENZ$(I)=V$:RETURN
415 Z$(I)="Z"+MID$(STR$(ZZ),2):Z$=Z$(I):RETURN
420 IFE<256THENGOSUB3B0:P$=Z$:RETURN
425 B=-1:H=PP+1
430 I=INT((N+B)/2):IFP(I)=ETHENP$=P$(I):RETURN
435 IFP(I)>ETHENH=I:GOTO445
440 B=I
445 IFABS(H-B)>1THEN430
450 IFPP>PMTHENP$="":RETURN
455 PP=PP+1:IFP(I)<ETHENI=I+1
460 FDRB=PPTDI+1STEP-1:P(B)=P(B-1):P$(B)=P$(B-1):NEXT:P(I)=E
462 IFS=2THENP$(I)=V$:RETURN
465 P$(I)="W"+MID$(STR$(PP),2):P$=P$(I):RETURN
470 DATABRK,ORA9,?,?,?,ORA2,ASL2,?,PHP,ORA1,ASL12,?,?,ORA5,ASL5,?,BPL8
471 DATAORA10
475 DATA?,?,?,ORA3,ASL3,?,CLC,ORA7,?,?,?,ORA6,ASL6,?,JSR13,AND9,?
480 DATA?,BIT2,AND2,ROL2,?,PLP,AND1,ROL12,?,BIT5,AND5,ROL5,?,BMI8,AND10
481 DATA?,?,?
485 DATAAND3,ROL3,?,SEC,AND7,?,?,?,AND6,ROL6,?,RTI,EOR9,?,?,?,EOR2
486 DATALSR2
490 DATA?,PHA,EOR1,LSR12,?,JMP13,EOR5,LSR5,?,BVC8,EOR10,?,?,?,EOR3,LSR3
491 DATA?
495 DATACLI,EOR7,?,?,?,EOR6,LSR6,?,RTS,ADC9,?,?,?,ADC2,ROR2,?,PLA,ADC1
500 DATAROR12,?,JMP11,ADC5,ROR5,?,BVS8,ADC10,?,?,?,ADC3,ROR3,?,SEI
501 DATAADC7,?,?
505 DATA?,ADC6,ROR6,?,?,STA9,?,?,STY2,STA2,STX2,?,DEY,?,TXA,?,STY5,STA5
510 DATASTX5,?,BCC8,STA10,?,?,STY3,STA3,STX3,?,TYA,STA7,TXS,?,?,STA6,?
511 DATA?
515 DATALDY1,LDA9,LDX1,?,LDY2,LDA2,LDX2,?,TAY,LDA1,TAX,?,LDY5,LDA5
516 DATALDX5,?
520 DATABCS8,LDA10,?,?,LDY3,LDA3,LDX4,?,CLV,LDA7,TSX,?,LDY6,LDA6,LDX7,
525 DATACPY1,CMP9,?,?,CPY2,CMP2,DEC2,?,INY,CMP1,DEX,?,CPY5,CMP5,DEC5,?
530 DATABNE8,CMP10,?,?,?,CMP3,DEC3,?,CLD,CMP7,?,?,?,CMP6,DEC6,?,CPX1
531 DATASBC9
535 DATA?,?,CPX2,SBC2,INC2,?,INX,SBC1,NOP,?,CPX5,SBC5,INC5,?,BEQ8,SBC10
536 DATA?,?
540 DATA?,SBC3,INC3,?,SED,SBC7,?,?,?,SBC6,INC6,?
545 FORI=0TOLL:IFV$<>""THENIFLEFT$(L$(I),1)<>V$THEN555
550 E=L(I):GOSUB300:PRINTL$(I)SPC(8-LEN(L$(I)))E$
555 NEXT:PRINT:GOTO45
560 FORI=0TOZZ:E=Z%(I):GOSUB300
565 PRINTZ$(I)SPC(8-LEN(Z$(I)))E$:GDTO555
570 FORI=0TOPP:E=P(I):GOSUB300
575 PRINTP$(I)SPC(8-LEN(P$(I)))E$:GDTO555
600 S=2:INPUT"NAME,AD";V$,E$:IFV$<>"END"THENGOSUB280:GOSUB420:GOTO600
605 GOTO45
1000 FORI=14TO33:PRINTICHR$(34)CHR$(I)CHR$(34);:NEXT
1001 END
```

MICRO

The A2-GE1 Graphics Editor
for the Apple II

The A2-GE1 Graphics Editor is a collection of programs designed to put the power of A2-3D1 and A2-3D2 graphics in your hands.

The A2-GE1 includes **Object Editor, Motion Programmer, Motion Playback, Slideshow Playback**, and a special A2-3D2 interface for BASIC programmers.

With **Object Editor** you can create whatever objects you want in the colors of your choice. You can also type in whatever 3D text you want, and in different sizes. And saving an object is as easy as naming it.

Then give the object names to **Motion Programmer** and see how the beautifully laid out keyboard controls will let you switch objects on or off, animate them, or add upper or lower case 2D text mixed right in.

You can also record your entire presentation, animation and all, for later use with **Motion Playback**, or just take "computer snapshots" of scenes with **Slide Show Playback**.

We've reached our goal of giving you the most sophisticated graphics utilities in the marketplace . . .

See them now at your dealer!

Convenient graphics power...

A2-GE1 Graphics Editor
$34.95 on disk (48K and A2-3D2 required)

A2-3D1 with 3D2 Enhancement*
$84.90 on disk (48K required)

*3D1 owners may update to 3D2 for $24.95. Contact SubLOGIC for details.

For direct order, include $3 for UPS or $5 for first class mail delivery. Illinois residents add 5% sales tax. Visa and MasterCard accepted

subLOGIC
Communications Corp.
713 Edgebrook Drive
Champaign, IL 61820
(217) 359-8482
Telex: 206995

Pascal Tutorial

Part 2

Victor R. Fricke
325 Ramapo Valley Road
Mahwah, New Jersey 07430

Filer Revisited

The Apple Pascal Filer is a system program whose primary function is to manage and control the interaction of the system with disk files. One particular file is handled in a special way by the system. This special file is the "workfile." When you Q(uit a session with the Editor, the Filer makes a copy of the text you were working on. This copy is placed on the disk as a TEXT file, with the file name of SYSTEM. WRK.TEXT.

The Editor scans the disk directory of the disk from which you booted the system. If it sees a program called SYSTEM.WRK.TEXT, it automatically reads it from the disk and displays it on the screen. The Editor assumes that you want to work on the workfile. But what if you want to stop working on one file and start working on another? The Filer is provided with two commands for this very purpose. C(hange can be used to change the name of SYSTEM.WRK. TEXT, and G(et can be used to designate a different file as the workfile.

C(hange

Suppose you have been working on a business accounting package. You have just successfully compiled a General Ledger program, and now want to turn your attention to writing the Accounts Payable program. You need to save the GL program under a different name, like LEDGER.TEXT. There are already copies on the disk of both the text and code files for this program, but they are named SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE.

You could, of course, go back to the Editor and then Q(uit by using the W(rite option, but this is not very efficient. First, the Editor would read SYSTEM.WRK.TEXT back into memory when you invoke it, even though it is already there. Second, SYSTEM.WRK.CODE would not be renamed by this process.

A better method would be to go into the filer, and select the C(hange option. The prompt line will say:

CHANGE? Respond by typing SYSTEM.WRK.TEXT

The system answers with:

CHANGE TO WHAT? You can now answer LEDGER.TEXT. The system response will be SYSTEM.WRK.TEXT ->LEDGER.TEXT.

After you do this, there is no longer a file named SYSTEM.WRK.TEXT in the disk directory. The file is still there, but its name has been changed. Now when you enter the Editor, there is no workfile to be automatically read from the disk. You can start a new one, and when you Q(uit, it will be saved as SYSTEM. WRK.TEXT.

The previous sequence of commands shows how to change a file name. This can be made even simpler. The two responses given above can be made at the same time, if they are separated by a comma. Thus, in response to CHANGE? you can type SYSTEM.WRK.TEXT,LEDGER.TEXT and get the same result.

If you want to change several files with similar names, you can use the "wild card" characters, '=' and '?'. Suppose your diskette contained the following files:

SUPER.LEDGER.TEXT
SUPER.PAYABLE.TEXT
SUPER.RCVABLE.TEXT
SUPER.STARTREK

If you wanted to rename all these files without the SUPER prefix, you could C(hange each name, one at a time. However, there is an easier way; use the "wild card."

From the Filer press C(hange. When asked CHANGE? type SUPER.=,= and the response will be

SUPER.LEDGER.TEXT -> LEDGER.TEXT

SUPER.PAYABLE.TEXT -> PAYABLE.TEXT

SUPER.RCVABLE.TEXT -> RCVABLE.TEXT

SUPER.STARTREK -> STARTREK

The system selects those files that have the prefix SUPER. and any suffix (represented by the wild card character) and changes its name to just the suffix.

If you wanted to change the name of all the accounting files, but leave SUPER. STARTREK alone, you could have followed one of two options. You could have answered the prompt with

SUPER.=.TEXT,=.TEXT

or,

SUPER.?,?

When '?' is used instead of '=', the system stops before each file name change and requests verification that the change of name is desired for that file. It will prompt with

CHANGE SUPER.LEDGER.TEXT ?

If you respond by pressing 'Y', the response will be

SUPER.LEDGER.TEXT --➤
LEDGER.TEXT

If you press any other key, the change will not be made, and the system will continue looking for more file names to change according to your instructions. In this way you can examine each name change before it is made and select those which you really want to change.

When there is no workfile defined, or you have just renamed SYSTEM. WRK.TEXT, the system is not able to automatically load the workfile. If you want to designate a different file as the workfile, select the Filer command G(et. The response will be

GET ?

You then respond with the name of the file you want to be the workfile. But there is one trick on a one-disk drive system like mine. The file you select by the G(et command has to be physically on the system diskette (the one with the system programs on it). If it is on another diskette, transfer it to the system diskette before using the G(et command.

The N(ew command is used to delete the current workfile. If there is a SYSTEM.WRK.TEXT or SYSTEM. WRK.CODE file on the disk, the response to the N(ew command is

THROW AWAY CURRENT
WORKFILE?

If you answer 'Y', the workfiles are removed from memory and the disk directory. If you press any other key, the N(ew command is cancelled. If there is no SYSTEM.WRK.TEXT or CODE file and you have designated a workfile by the G(et command, the N(ew command de-designates it, but does not remove it from the disk.

To find out what the system configuration is, The V(olumes command is used. This command is inherited from the UCSD System, which on occasion is run on large computers, small computers, time-sharing computers, or other hardware. The system software is set up to deal with all the possible hardware variations by regarding each device it can communicate with as a volume.

When you select the V(olumes command, the display, for a single drive system, will look like this:

```
VOLS ON-LINE:
1     CONSOLE:
2     SYSTERM:
4 #   APPLE0:
ROOT VOL IS - APPLE0:
PREFIX IS     - APPLE0:
```

The volume number designations, since they were set up for the UCSD system, do not correspond to peripheral slot numbers. For example, Volume 4 represents the on-line disk in drive 1, in slot 6.

The volume designated as CONSOLE: (Volume 1) is the video display; Volume 2, SYSTERM:, is the keyboard. A '#' symbol in front of the volume name indicates that it is a "block-structured" volume, or diskette.

The "ROOT VOL" referred to is the volume from which the system software was booted up. The "PREFIX" is the name of the default volume; that is, the volume that is assumed when only the file name is given to the system. This is ordinarily the same as the root volume, but can be changed by using the P(refix command.

For further information about system volume numbers and what peripheral slots they can represent, refer to Appendix D in the back of the *Operating System Manual*.

Directory Commands

There are several commands which access the directory or modify its contents.

Z(ero — Wipes out the directory. Programs are still on the disk, but the system can't find them because the directory is empty. The only time this might be of any use is if you want to re-use an old disk.

R(emove — Used to delete a file from the directory. The file is still on the disk, but after a R(emove, the system thinks that the area where the file is recorded is available for use. Subsequent file creation can wipe out the file just R(emoved.

K(runch — This command is used to move the existing files together on the disk, making all the remaining unused space on the disk contiguous. Since this method involves reading files and writing them elsewhere on the disk, this command should be used sparingly. A disk error or power failure during a K(runch operation could cause permanent loss of some files.

M(ake — This command is used to create a dummy file and put its name into the directory. The only reason I can see for doing this is to try to recover an inadvertantly R(emoved file. For example, I R(emove a file called JUNK.TEXT. I realize that I have just killed my only copy of a useful file by that name. When I use the E(xtended directory list command, I see an area of nine blocks between two files marked <UNUSED> I think that might be where JUNK.TEXT is recorded.

I use the M(ake command and give it JUNK.TEXT[9] as the name of the new file. The 9 in the square brackets is the number of blocks to allocate for the file. This procedure will recover the lost file.

If you use the M(ake command and use a file name without the size specified in square brackets, the new dummy file will still be made, and it will fill the largest unused area. If you use an asterisk in the square brackets, the new file will occupy either one half of the largest unused area, or the next-to-largest unused area, whichever is larger.

Bad Disk Blocks

The system provides a command, B(ad blocks, which instructs it to scan the disk for flaws and identify them. This involves a "CRC," or "cyclic redundancy checksum." When a block of data is recorded on the disk, the CRC is calculated and stored in the sector along with the data.

When the B(ad blocks command is selected, the system reads each sector, calculates the checksum for the data in that sector, and then compares the result with the CRC stored on the disk. If after ten attempts, no match occurs between the calculated and recorded checksums, the system concludes that a bad block has been found.

Make note of the bad blocks found by the scan. Although they cannot be fixed, the system can mark them as bad. You invoke this operation by using the X(amine command. This causes the system to mark the bad blocks as a file with the suffix .BAD. This is important because in a K(runch, the system will not attempt to move any portion of a file into the bad block.

Unfortunately, the file which contained the bad block will not be recovered, but at least you will not jeopardize any other good files. It is a good idea to use the B(ad blocks and X(amine commands right away when you first initialize a diskette for Pascal files.

Compiler

The Pascal Compiler is a translation program. It translates a Pascal text into a different language, "p-code." The Apple is a willing servant, and will carry out any instruction it receives, as long as it can understand the instruction. Of course, the only instructions it understands are those written in 6502 machine language.

Normally, humans do not communicate in machine language; it is hard to deal with binary bytes. The closest one usually comes is assembly language. But fortunately, the Apple monitor understands assembly language, and is prepared to interpret it to the 6502 CPU. The mini-assembler system program does this.

Carrying the analogy a step further, assembly is not a comfortable language for the average programmer. Most would rather speak BASIC, since it resembles English. Fortunately, the Apple also has a BASIC interpreter.

An interpreted program is inherently slower-running than it could be, since each statement is decoded as it is encountered. If a loop is executed a thousand times, the loop statements are decoded a thousand times. It would be much more efficient to do the decoding only once, translate into machine code, and then run the translated program.

A compiler takes a high-level language program, understandable by a human, and translates it into a low-level language that the CPU understands. The advantage is that the low-level language is more concise, and therefore uses less valuable memory space, and runs faster. The disadvantage is that if you make one small change, the program must be recompiled. This is usually not much of a problem, except during program development.

Now Pascal, a new language designed from scratch to have a lot of desirable features, eliminates a lot of the weaknesses and faults of existing languages. The designers wanted it to be a compiled language, but they faced a problem: different computers spoke different machine languages.

The Apple speaks 6502, the TRS-80 Color Computer speaks 6809, and other machine languages for other processors abound. A compiler is a very complex program, and writing a Pascal compiler for each of the possible processors multiplied the problem.

The solution was to invent a new low-level language, "p-code," and to write only one compiler, Pascal to "p-code," the *universal* low-level language. Now, to use the Pascal system software on a new microprocessor, only a "p-code" interpreter need be written. Interpreters are much easier to write than compilers.

A Pascal program is a text file on the disk. All the compiler does is read the text file, translate it into "p-code," and write a code file onto the disk. Very little interaction with the user is needed for this type of operation. However, there are a few options available. Rather than using a prompt line approach, the compiler options are se-

lected by means of compiler directives embedded in the text file.

The compiler directive looks like a comment. A Pascal comment is enclosed by parentheses and asterisks:

(* THIS IS A COMMENT *)

Comments are ignored by the compiler. But, compiler directives are not because they start with (*$, not just with (*. Compiler directives look like this:

(*$S+ *)   (*$L– *)

There are really only three compiler directives that are of any use to the beginner. These are the swapping directive, the list directive, and the include directive.

The swapping directive is used to conserve memory working space. The compiler is held in memory while it is operating. If you look at a directory listing, you can see that SYSTEM. COMPILER occupies 71 blocks of 512 bytes each, or over 35K of the available memory space.

The swapping directive tells the system to divide the compiler into two parts, and to swap the parts in and out of memory. Only the part needed is in memory at any time. This frees additional memory space and allows the compilation of larger, more complex programs.

```
 1   1   1:D    1 (*$L CONSOLE:   *)
 2   1   1:D    1 PROGRAM SUMS;
 3   1   1:D    3
 4   1   1:D    3 CONST
 5   1   1:D    3    MAXINT = 100;
 6   1   1:D    3
 7   1   1:D    3 VAR
 8   1   1:D    3    SUM :   REAL;
 9   1   1:D    5    COUNT:INTEGER;
10   1   1:D    6
11   1   1:D    6
12   1   1:0    0 BEGIN
13   1   1:1    0    SUM := 0;
14   1   1:1    8    FOR COUNT := 1 TO MAXINT DO
15   1   1:2   19       SUM := SUM+COUNT;
16   1   1:2   37
17   1   1:1   37    WRITELN('SUM= ', SUM)
18   1   1:1   77
19   1   1:0   77 END.
```

This option is highly recommended for development of programs of any useful size. It is invoked by placing the compiler directive

(*$S+ *)

at the very beginning of the text file. You may place comments before it in the file, of course, since comments are ignored by the compiler.

When the compiler is running, you get very little information about its progress. A dot is placed on the screen as each line of text is compiled, and the names of procedures and functions appear as their compilations start. The remaining space in memory appears in square brackets. When an error is detected, the compilation stops, and the line number where the error was detected is indicated.

The compiler translates the program one line at a time, and it numbers the lines for reference purposes. It would be useful to see the numbered lines as the compilation progresses, just so you could fix the errors more easily. The list directive does this for you. If you include

(*$L CONSOLE: *)

at the head of your program, you will get a detailed listing on the screen as the compilation progresses. It will look like listing 1.

The numbers in the first column are the line numbers. In the second column the segment number is indicated. Segment numbers are of use only to the advanced Pascal programmer. In ordinary programs, the segment number will always be "1".

The third column contains two numbers, separated by a colon. The first number is the block number. Each procedure, function, and the main program are blocks, separately named. The second number is the indentation level. A 'D' means declaration; the numbers show the level of nesting of loops and other control structure in the block. The final column of numbers is intended for use with a debugger subsystem, and is of little use for the beginning programmer.

The compiler directive (*$I FILENAME *) is called the Include directive. It causes the compiler to fetch additional text from the file named FILENAME and insert it into the textfile being compiled. In this way, you can break a large file into smaller pieces and work on them separately. As an example, you could use a procedure from another program that is already known to work.

MICRO

Flags and Boolean Algebra in Microsoft BASICs

Microsoft BASICs, unlike other BASICs, can handle the assignment of Boolean variables. Furthermore, OSI and PET versions work on the bit level. Both features allow implementation of some very powerful program structures.

Mark Guzdial
1451 Seminole
Royal Oak, Michigan 48071

Flags are variables that have only two values: a 'true' and a 'false'. They're heavily used in languages other than BASIC, such as PL/1 and FORTRAN, but this doesn't mean that they can't be used in BASIC and with practical benefit.

For example, a flag named ER could be set true upon an error condition. A statement using it might be 'IF ER THEN PRINT ER$' thus printing an error message upon an error condition. Or, a flag called LP could indicate the presence of a line printer, so a print message might be 'PRINT A$:IF LP THEN LPRINT A$'.

Flags can be put to their greatest use by combining them with Boolean algebra (as discussed by Marvin DeJong in MICRO 22:29). In some other languages, flag variables are referred to as Boolean. For example, we can combine our last two examples by printing an error message to the printer with 'IF LP AND ER THEN LPRINT ER$'.

In BASIC, if the flag is non-zero (a positive or negative number), it will be considered true and the statement after the THEN will be executed. If the flag is zero, it will be considered false and the statement after the THEN will be skipped.

What really makes flags usable in Microsoft BASIC is that BASIC can actually handle the assignment of true and false values. For example, the statement A=B<2 is actually a valid statement in any Microsoft BASIC (including PET BASIC, OSI, Applesoft, and even Apple Integer BASIC, though it wasn't written by Microsoft).

The explanation lies in the structure of BASIC. Discounting strings, there are two main types of expressions: arithmetic (those using addition, subtraction, functions and generally working with numbers) and logical (those that make comparisons such as =, < or >). In PET, OSI and Apple BASICs, the evaluation of these expressions is done by the same routine. That means that the expression between the IF and the THEN is evaluated the same way as the expression to the right of the '=' sign in the assignment statement. Therefore A=B+2 is just as valid as A=B<2.

We can now also see what happens if we try the statement A=B=0. The variable A will be set to true or false depending on whether or not B is equal to 0.

Let's digress slightly from our discussion of flag variable usage to discuss this evaluation routine. We know that BASIC can only put a number into the variable A, so what number does BASIC put into A when we use it to signify a true or false condition such as A=B<2?

If B is not less than 2, A will be set to 0 in all of these BASICs. From what we know about false conditions, this sounds correct. But it's true values that get tricky. If we consider Boolean algebra, true should be the complement of false, or TRUE=NOT FALSE. So, logically, if B is less than 2 (using our example), A should get the value of 1 — the complement of 0.

In the Apple, both Integer and Applesoft BASICs return 1 for true values. But on OSI and PET BASICs, A will be set to −1! And believe it or not, this is a distinct advantage, though not a logical one.

In PET and OSI, flags and Boolean algebra are considered at a 'bit level' not as their logical values. This means that PET and OSI consider all numbers in terms of their binary digits. So the complement of 0 is:

NOT 0000 0000 = 1111 1111

For those of you familiar with two's complement notation (a method of representing negative numbers in machine code), you will recognize that the NOT of 0 is −1. A NOT of −1 would give you back the 0 which is correct. Also, since −1 is non-zero, it will be recognized as true so it will function correctly.

Note that while the Apple format can handle a −1 as being true (since it is non-zero), the OSI and PET formats cannot handle a 1 as being true if you're using Boolean algebra. A NOT of 1 would give you (in PET or OSI):

NOT 0000 0001 = 1111 1110

which is a two's complement −2, which (since it's non-zero) is still true.

This can be a distinct advantage. Though logically it's considerably easier to work with the Apple method of handling true and false, the bit-oriented way of the PET and OSI gives you another dimension of programming. For example, what is printed with the statement:

'PRINT 2 OR 4'?

In the Apple, you are logically ORing two non-zero (or true) values which leaves a true result, so a 1 is printed. However, on the PET or OSI a 6 is printed because

```
        0000 0010
   OR   0000 0100
        0000 0110 = Decimal 6
```

This leaves us with the capacity of doing some very interesting computations with the PET or OSI computers.

Let's say, for example, that the variable A must be assigned a value based on variable B, but the relationship is totally illogical such as:

```
IF B=1 THEN A=5
IF B=2 THEN A=3
IF B=3 THEN A=7
```

One's first thought would probably be to either use multiple IF..THENs or an ON..GOTO. But by using flags, this assignment can be done on a single statement:

```
A= -(((B=1)*5) OR ((B=2)*3)
OR ((B=3)*7))
```

We know that each of the conditional statements (B=n) will be evaluated to either -1 or 0, depending on whether it's true or false. Only one of the three clauses can be true, so only one can evaluate to a number while the other two evaluate to zeros. A number logically ORed with two zeros leaves the number. The negative sign at the beginning is necessary since B=n will be evaluated to a negative 1 if true.

This type of statement cannot be used on the Apple because the statement will logically evaluate the ORs to either a 1 or 0, not the number that will be found.

Using Boolean algebra with flag variables opens up a whole new world of programming in BASIC. Complex IF.. THENs can be broken down, IF..THEN ..ELSE structures can be implemented and sometimes, IF..THENs can be avoided completely. I hope that you find the use of flag variables in 6502 BASICs to be beneficial to your programming in BASIC.

MICRO

SPEED POWER EFFICIENCY for OSI

65D3 SYSTEMS

FBASIC: BASIC Compiler $155/$10

- FAST efficient machine code written with the ease of BASIC.
- SPEED-optimized, native-code compiler.
- INTEGER subset of OSI-supplied BASIC.
- DISK based: No problem with size of source or object files.
- EXTENSIONS to BASIC:
  - Simple interface to system hardware and software.
  - Direct access to 6502 registers.
  - Array initialization.
  - Optional absolute array locations.
  - WHILE and other structures.
  - Simple technique for combining compiler and interpreter advantages.
- FULL system: utilities (plus source), manual, and many useful examples.

R-EDIT: Edit any program or text with ease! $40

- FULL CURSOR control. Edit anywhere on the screen.
- INSERT, replace, add, delete.
- ONE KEY stroke and you're editing.
- BASIC, assembler, etc. can all be edited without reloading editor.
- RAM-resident: Always ready!
- SYSGEN relocates R-EDIT and customizes.

SPUL-65: Printer Spooler Virtual Indirect File $95/$10

- STOP WAITING for your printer!
- PROCESS words, write programs...all while printing!
- QUEUE lets you pile on print jobs.
- MULTIPLE COPIES printed with top and bottom page margins.
- SYSGEN relocates SPUL65 and gives extensive customization.
- INDIRECT FILE commands produce disk files giving you:
  - A virtually unlimited temporary file.
  - A link between incompatible files; for example, use WP-2 for extensive BASIC editing.
  - Ability to merge multiple program segments.

XREF: BASIC Cross Referencer $25

- TABULATES:
  - Referenced line numbers.
  - Variable names (numeric, string, array).
  - Defined functions.
- FAST machine language program.
- DISK based: Handles large BASIC source files on any drive.

CP/M to OSI Disk Translation

Frustrated by all those good CP/M disks that won't run on your OSI CP/M system? It's that special OSI disk format! And we can fix that. Just send us your disk, $15, and you'll soon have an OSI compatible disk.

drc software

Data Resource Corporation
Suite 204
1040 Lunaai Street
Kailua, HI 96734 (808) 261-2012

Manual orders applied to software purchases. Programs supplied on 8-in, single-density, single-sided disks. Hawaii residents add 4% tax.

Recursive Use of GOSUB in Microsoft BASIC

The concept of recursion, or repeatedly calling a routine from within itself, can make the coding of an algorithm much more efficient and elegant. Here, the use of recursion is explored within a microsoft BASIC environment. The OSI implementation discussed here is portable to Apple, PET, TRS-80 Color, and other microsoft BASIC computers.

Rolf B. Johannesen
13917 Congress Drive
Rockville, Maryland 20853

A subroutine is a group of statements which may be called repeatedly by a main program or by another subroutine. If it is called only once it should not be a subroutine, but rather, coded in line. A subroutine may have no arguments (e.g. a subroutine to generate carriage return — line feed), or may have one or more arguments which are changed with each call (e.g. SIN (x)).

For the subroutine to return to the calling program after it has finished, it must be given a return address by the calling program. This return address must be saved until it is needed. The mechanism of doing this is provided in the particular machine used, down to the hardware level, and is of concern to the programmer only under the following condition: if the return address is stored in a fixed location, a second call to the subroutine (before it has finished its task) will cause the return address to be overwritten and an eventual return to the original calling program will become impossible — disaster! However, if the return address is stored on a stack, then subsequent calls to the subroutine will cause the return addresses to be stored in successive locations on the stack and an orderly return back to the first level is possible.

Successive Calls: Recursion

Two questions present themselves immediately: why would you want to call a subroutine again before it has finished its task? And, if convinced that this was desirable, how can it be done?

To answer these questions in reverse order, a subroutine may *call itself* in certain circumstances. A skeleton example will illustrate this point:

```
 10 get data, set up parameters
 20 GOSUB 100
 30 output results
 40 either STOP, or GO to 10
100 If (condition) THEN (state-
    ment): RETURN
110 GOSUB 100
120 process results
130 RETURN
```

This skeleton can readily be developed into a workable program in such languages as ALGOL, Pascal, and, of particular interest here, Microsoft (OSI, Apple, PET, SYM, TRS-80 Color) BASIC. The ability of a subroutine to call itself is called *recursion* and is allowed in the languages mentioned above. Recursive calls, as in line 110 above, are explicitly disallowed in all versions of FORTRAN that I know of.

Example Programs Using Recursion

Probably the classic example of a recursive subroutine is that for calculating N factorial. To refresh your memory, N factorial, written N!, is defined for positive integers as N (N-1) (N-2) ... 1. 0! = 1. When N < 0, or N non-integral, then N! is undefined. Listing 1 illustrates the use of recursion for calculating N!. The subroutine begins on line 100. You can see that this example meets all the conditions given in the skeleton listing above. By running this program with various values of N as input, you'll see that an OM (out of memory) error occurs on the OSI when N = 25. Yet, PRINT FRE (x) gives a large positive number. The conclusion to be drawn from this is that stack overflow occurs after 24 recursive calls, even though there is really plenty of unused memory.

The program in listing 1 is, admittedly, a poor way to calculate factorials when a FOR-NEXT loop would be much easier. It is included solely to illustrate the use of a recursive subroutine in a way that is simple to trace through. Listing 2 gives a program which generates all possible permutations of a set of characters that are read in. (This listing illustrates the genesis of this article. I wanted to generate all possible anagrams from a given set of letters — the resulting program is given in listing 2.) In this case, recursion represents an easy way to generate the required permutations for any size array. If the size of the array is exactly three, all permutations are generated and printed in subroutine 590 without recursion. If the size of the array is four, then four calls to subroutine 590 are generated. For instance, if the original array is ABCD, the four sets of results are A(BCD), B(ACD), C(ABD) and D(ABC) where (ABC) represents all six possible permutations of A, B, and C. If the array size is larger than four, the subroutine works similarly. That is, in each step the size of the array is reduced by one and the subroutine is called again recursively. Figure 1 gives the results of a permutation of four characters. In the first example, all four are distinct, while in the second case there is one set of duplicate letters and the number of permutations is therefore cut in half.

The principle that is illustrated in each of these listings is generally valid: the recursive subroutine examines some

variable. If it is small enough (N = 1 in listing 1, or N = 3 in listing 2), then the solution is calculated at once. If the variable examined is too large to allow immediate solution, then it is reduced by one and the subroutine calls itself recursively. Upon return, the variable is incremented again and any necessary calculations are done on the returned values.

Listing 2 illustrates the use of a *software stack* in lines 380-400 and 480-510. It is necessary to generate the equivalent of a FOR...NEXT loop in cases where N > 3. However, a simple loop cannot be used or the loop variable would be written over when the subroutine was called recursively. Hence, we assign a variable stack pointer SP and use arrays NN and NS to hold the loop counters and the final values they are tested against respectively. If you follow the code you'll see that if N = 3, SP, NN and NS are never used. If N = 4, SP = 1, and in general SP = N - 3 where N is the original number of characters read in. To produce all possible permutations of seven characters requires about 13½ minutes on my OSI C1P machine. I have entered an array of eight characters and the results appear to start out correctly, although I have not waited for the run to finish.

The conversion from ASCII in line 80 should not be required. However, the initial version of this program used A$ as a character array exactly as A is used in listing 2. Unfortunately, the program would only run for a few cycles before breaking down and filling my original array with repetitions of the same character. This behavior is probably related to the well-known string array bug in this machine. In any case, the present version appears to be bug-free for strings up to at least eight characters in length.

Rolf B. Johannesen has worked as a chemist at the National Bureau of Standards in Washington, D.C. and Gaithersburg, Maryland since 1951. He was introduced to FORTRAN programming in 1965, and has since programmed in many languages, both high-level and assembly. He has a KIM-1 and an OSI C1P with 8K BASIC in ROM plus 8K RAM.

Editor's note: This article is not affiliated with Mr. Johannesen's place of employment.

Listing 1

```
10 INPUT "ENTER N";N
20 IF N<0 OR INT(N)<N THEN 60
30 GOSUB 100
40 PRINT N;"FACTORIAL =";F
50 GOTO 10
60 PRINT N;"FACTORIAL UNDEFINED"
70 GOTO 10
100 IF N=1 OR N=0 THEN F=1:RETURN
110 N=N-1
120 GOSUB 100
130 N=N+1
140 F=F*N
150 RETURN
```

Listing 2

```
10 REM PERMUTATION ROUTINE
20 REM ILLUSTRATES RECURSIVE USE OF GOSUB
30 REM AUTHOR  -  ROLF B. JOHANNESEN
40 REM LAST REVISION 20 JAN 81
50 INPUT W$
60 L=LEN(W$)
70 FOR I=1 TO L
80 A(I)=ASC(MID$(W$,I,1))
90 NEXT
100 REM PERMUTE NUMERIC VALUES OF CHARACTERS
110 REM IF STRING LENGTH>10, A MUST BE DIMENSIONED
120 REM (NOTE: THERE ARE 3628800 PERMUTATIONS OF
130 REM TEN DIFFERENT OBJECTS!)
140 N=L
150 REM INITIALIZE POINTERS AND COUNTERS
160 SP=0:PS=0:K=0
170 GOSUB 870:REM SORT AARAY INTO ASCENDING ORDER
180 GOSUB 270:REM FIND THE PERMUTATIONS
190 REM IF ONE OR TWO PERMUTATIONS LEFT OVER,
    PRINT BEFORE GETTING NEXT
200 ON PS GOTO 220,230
210 GOTO 50:REM GET NEXT STRING
220 PRINT TAB(2);X$:GOTO 50
230 PRINT TAB(2);X$;SPC(1);Y$:GOTO 50
240 REM
250 REM
260 REM
270 REM  SUBROUTINE PERMUTE
280 REM THIS SUBROUTINE DOES THE WORK
290 REM STARTING FROM THE RIGHT, TRANSPOSE AND
300 REM WORK TO LEFT UNTIL ALL POSSIBLE TRANSPOSITIONS
310 REM (PERMUTATIONS) HAVE BEEN DONE
320 IF N=3 THEN GOSUB  590 : RETURN
330 REM PERMUTE 3 OBJECTS AND PRINT
340 REM SORT INTO ASCENDING ORDER
350 GOSUB 870
360 REM IF N>3 THEN INCREMENT STACK POINTER (SP)
370 REM COUNT DOWN ON N, AND CALL PERMUTE AGAIN
380 SP=SP+1
390 NS(SP)=N:REM HERE IS A FOR-NEXT LOOP
400 NN(SP)=1:REM WITH NN AS LOOP VARIABLE
410 N=N-1:REM AND NS AS FINAL VALUE
420 GOSUB 270:REM RECURSIVE CALL WITH N DECREMENTED BY ONE
430 REM DO NOT PERMUTE IDENTICAL OBJECTS
440 IF A(L-N)=A(L-NN(SP)+1) THEN 540
450 T=A(L-N):A(L-N)=A(L-NN(SP)+1):A(L-NN(SP)+1)=T
460 REM SWAP ABOVE IS NEXT STEP IN PERMUTATION
470 REM AFTER RIGHTMOST GROUP OF 3 HAS BEEN PERMUTED
480 IF NN(SP)<NS(SP)THEN NN(SP)=NN(SP)+1:GOTO 420
490 SP=SP-1
500 REM BACK UP THE STACK
510 N=N+1:REM AND COUNT UP ON N
520 RETURN:REM REACHES HERE EVENTUALLY
530 REM NEXT INCREMENT LOOP COUNTER IF AN IDENTICAL
    PAIR IS FOUND
```

(Continued)

Listing 1 *(Continued)*

```
540 IF NN(SP)<NS(SP) THEN NN(SP)=NN(SP)+1
    :GOTO 440
550 GOTO 490
560 REM
570 REM
580 REM
590 REM  SUBROUTINE PERMUTE3
600 REM WHEN N=3 THIS ROUTINE IS CALLED
610 GOSUB 870:REM SORT 3 IN ASCENDING ORDER
620 REM SET UP PRINT OF CURRENT PERMUTATION
630 W$=CHR$(A(1))

640 FOR I=2 TO L
650 W$=W$+CHR$(A(I))
660 NEXT I
670 ON PS GOTO 700,710
680 REM FALLS THROUGH IF PS=0
690 X$=W$:GOTO 730
700 Y$=W$:GOTO 730
710 PRINT TAB(2);X$;SPC(1);Y$;SPC(1);W$
720 PS=FRE(0):PS=-1
730 PS=PS+1
740 REM CHECK IF PERMUTE IS FINISHED
750 IF A(L-2)=>A(L-1) AND A(L-1)=>A(L) THEN 830
760 IF A(L)<=A(L-1)THEN 790
770 T=A(L):A(L)=A(L-1):A(L-1)=T
780 GOTO 630
790 IF K=2 THEN 830
800 IF A(L-2)=A(L-K) THEN K=K+1:GOTO 790
810 T=A(L-2):A(L-2)=A(L-K):A(L-K)=T
820 K=K+1:GOTO 770
830 K=0 : RETURN
840 REM
850 REM
860 REM
870 REM  SORT ROUTINE
880 REM SORTS LAST  N  ITEMS OF ARRAY  A
890 REM OF LENGTH  L  IN ASCENDING ORDER
900 FOR IA=L-N+1 TO L-1
910 S=A(IA)
920 IB=IA
930 FOR IC=IA+1 TO L
940 IF A(IC)=>S THEN 970
950 S=A(IC)
960 IB=IC
970 NEXT IC
980 IF IB=IA THEN 1000
990 A(IB)=A(IA):A(IA)=S
1000 NEXT IA
1010 RETURN
```

Figure 1

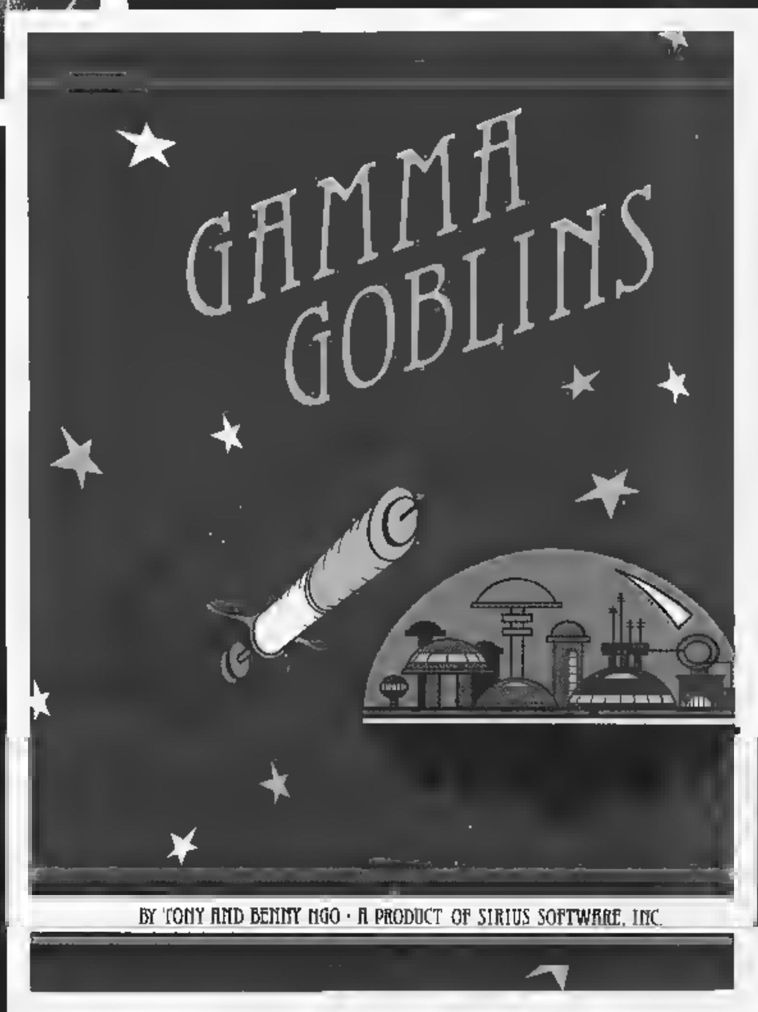
```
OK
RUN
? TRAP
  APRT APTR ARPT
  ARTP ATPR ATRP
  PART PATR PRAT
  PRTA PTAR PTRA
  RAPT RATP RPAT
  RPTA RTAP RTPA
  TAPR TARP TPAR
  TPRA TRAP TRPA
? DOOR
  DOOR DORO DROO
  ODOR ODRO OODR
  OORD ORDO OROD
  RDOO RODO ROOD
```

MICRO


LOPEZ 81


EPOCH

In the center of an orbiting space station you are protected only by a revolving force shield. Enemy forces are advancing from all directions and begin to place killer satellites in orbit around your station. And then, look out for the meteors!

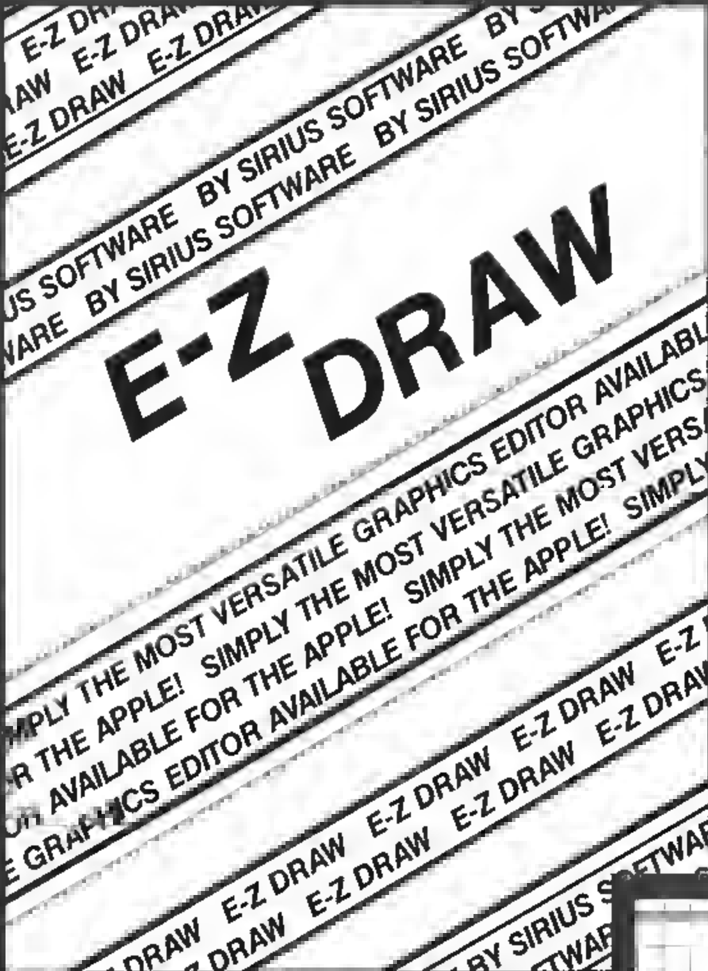


A "bloody" good game for the true-blue game freak. Your mission in this exploratory operation is to deliver whole blood to Hemophilia, a city in the sky, and return to Anemia Base before the Gamma Goblins overcome you. A real heart stopper!

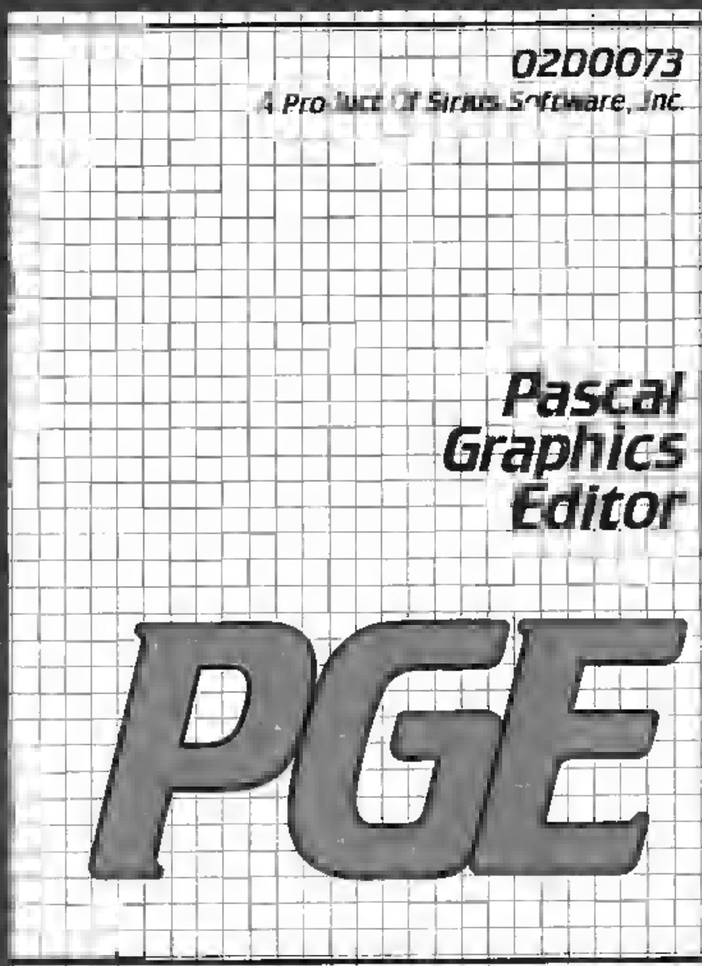

SNEAKERS

Phantoms Five simulates a fighter-bomber mission in real time, three dimensional color graphics. While you try to make your bombing run, you have to avoid being hit by anti-aircraft fire and fight off enemy aircraft as well.

Hatch some fun with the Spiders, Wolves, Lips, and Fuzzballs. Space Eggs will crack you up! Each package includes a multi-color T-shirt iron-on that says "I FRIED THE SPACE EGGS."



This is the graphics editing package we based our business on. Includes the Higher Text Character Generator by Ron & Darrell Aldrich and over 20 original and imaginative type styles.



The professional graphics editing package for use within the Pascal environment.

Short Subjects

Add a CALL Function to OSI ROM BASIC

Earl Morris
3200 Washington St.
Midland, Michigan 48640

Jim Cathey
Route 2 Box 468
La Center, Washington 98629

OSI BASIC-in-ROM does not have a command to jump directly to a machine language subroutine. Other micros have a statement such as SYS XXXX or CALL XXXX, where XXXX is the address of the desired subroutine. Machine language routines can be accessed through the USR function. However, this must first be set up with a POKE 11,X POKE 12,Y where X and Y are the decimal equivalents of the hexadecimal address. The conversion to decimal and POKEing two locations becomes tedious when several different machine routines are being accessed.

The following program will add a patch to ROM BASIC allowing commands of the form

Z= USR (1) ABCD

where ABCD is the hexadecimal address of your subroutine. The machine patch is located at $0240 below the start of BASIC. BASIC routines at $BC and $C2 are used to fetch the additional characters from the line of BASIC. A routine at $FE93, from the monitor, is used to convert ASCII data into a four-bit binary number. This routine also checks for invalid hex data. An invalid address like

Z= USR (1) ABCQ

will cause a return to BASIC with a "SN ERROR" message. The argument for the USR function must be present to avoid a syntax error, however it is not used here.

Once the patch is entered, the BASIC USR function must be initialized by

POKE 11,64 : POKE 12,2

The form of USR given above can now be used. Several example locations are:

Z= USR (1) 0000
Jumps back to BASIC with "OK"

Z= USR (1) FE00
Monitor

Z= USR (1) A4B5
Lists BASIC program

Z= USR (1) FF00
C/W/M ? Message

If you are putting BASIC into EPROM, the "NULL" command can be changed to a "CALL" command by changing the letters "NU" at $A0C5. The destination address at $A022 and $A023 must be changed to point to the routine listed here (minus one). If this change is made, BASIC will accept a command of the form

CALL FE00

and jump to the routine at $FE00.

Morris/Cathey Listing

```
               ;         ****************************************
               ;         * CODE TO EXECUTE USR(8) ABCD WHERE      *
               ;         *        'ABCD' IS HEX ADRESS OF USER SUB *
               ;         ****************************************
               ;
               ;          This code allows a user to execute a machine
               ;          language subroutine by address from a BASIC
               ;          program.
               ;
               ;          To use, the user must first set up the USR
               ;          vector by POKE 11,64 AND POKE t2.2.
               ;          Thereafter. the user may specify the address
               ;          of his ml sub after the USR call in BASIC.
               ;
               ;
0240                *    =     $0240
               ;
0240 20C200          JSR   $C2       GET CURRENT CHARACTER
0243 A201            LDX   #$01      SET COUNTER
0245 D003            BNE   J1        BRANCH ALWAYS
0247 20BC00 J2       JSR   $BC       GET NEXT CHARACTER
024A 2093FE J1       JSR   $FE93     MONITOR ROUTINE ASCII TO NIBBLE
024D 301B            BMI   RET       INVALID HEX
024F 0A              ASL   A         SHIFT 4 BITS LEFT
0250 0A              ASL   A
0251 0A              ASL   A
0252 0A              ASL   A
0253 9511            STA   $11.X     STORE PARTIAL ADDRESS
0255 20BC00          JSR   $BC       GET NEXT CHARACTER
0258 2093FE          JSR   $FE93     ASCII TO NIBBLE
025B 300D            BMI   RET       INVALID HEX
025D 1511            ORA   $11.X     COMBINE WITH PREVIOUS NIBBLE
025F 9511            STA   $11,X     STORE BOTH NIBBLES
0261 CA              DEX
0262 10E3            BPL   J2        IF X=1 GO AROUND AGAIN
0264 20BC00          JSR   $BC       GET NEXT CHARACTER
0267 6C1100          JMP   ($11)     JMP INDIRECT TO ROUTINE CALLED
026A 60     RET      RTS             INVALID HEX - RETURN TO BASIC
```

Plotting Figures from Applesoft

Harry L. Pruetz
2929 Clydedale #376
Dallas, Texas 75220

High-Resolution plots on the Apple II, for figures whose boundaries are not mathematical functions, may be implemented with extensive data statements in Applesoft BASIC programs. Smaller shapes may be stored in shape tables and displayed using the Applesoft 'DRAW' statement. A more practical approach is to use piecewise approximations to edges of figures and 'HPLOT' along the *x*-coordinate or *y*-coordinate.

The listed Applesoft BASIC program plots a figure often presented as an example of optical illusions and is sometimes used in psychological tests. Depending on how it is presented or who is observing it, the figure looks like a vase, or two opposed human profiles.

The coordinate definitions for Hi-Res points cause problems for most programs. Ideally, a program should be designed using coordinate conventions imposed by the application, with constants and sign changes delegated to a

subroutine. However, if you know at the start of program writing that you're going to use Hi-Res plots, the actual colors to be used and location of points are important. For example, a smooth and continuous plot of a function is difficult to achieve in any color except white. Even given that function limits are known and white is to be used, smooth plots are not guaranteed unless small step sizes are used along a coordinate. In the first three cases of the program, relative functions were used so that Hi-Res *y*-coordinates were used in the main program. The figure is symmetric, therefore the mirror image of one side is calculated in a subroutine.

The case loop extends from line 10 through 230. Each case is simply a different color scheme using the same figure. The figure was broken into fourteen separate pieces, P1 through P14. Liberal use of different mathematical functions was made for each section, although simpler functions were possible for smaller sections.

The routine from line 300 through 360 displays titles for each case. Lines 400 through 490 plot solid blocks above and below the vase figure.

The routine from line 500 through 575 calculates actual *x*-coordinates using the function defined outside the routine. Case 2 plots an entire white and blue horizontal line. Note the end-point changes necessary to prevent black spaces at the end-points.

Case 4 extends from line 235 through 290. In this example, *x*-coordinate values for a circle, and spirals within the circle, are multiplied by 1.0833 to give a better display on the CRT. Without this factor, the circle drawn by line 245 is obviously non-circular, no matter how much the television is adjusted. The factor 1.0833, or 13/12, is not exact, but is easily remembered. The factor appears again in lines 255 and 265. The point plotted at line 255 and erased at line 275 serves as an angle indicator as the spirals get denser. As this case is quite time-consuming, the bell is sounded repeatedly as soon as the plot is completed to alert those who have tired of watching. The screen is cleared when a key is pushed in response to the 'GET C$' at line 290.

The entire program is less than 2048 bytes long, which is comparable to the storage needed for the many data statements required to define the figure. Also, the programming was easier than a long list of data statements and probably contained fewer errors.

```
0  REM  ***********************
1  REM  *                     *
2  REM  *   FIGURE PLOTTER    *
3  REM  *       PRUETZ        *
4  REM  *                     *
5  REM  *     LISTING 1       *
6  REM  *                     *
7  REM  ***********************
8  REM


9 PI = 3.1415926536:XC = 140
10  FOR CASE = 1 TO 3: GOSUB 300: HGR2 : GOSUB 400: HCOLOR= 3
15  IF CASE = 1 THEN  HPLOT 70,10 TO 210,10
20 YA = 10:YB = YA + 15:KA = 70:KB = PI / 10:KC = 10
25  DEF  FN F(Y) = KA + 6 *  SIN (KB * (Y - KC))
30  GOSUB 500: REM P1
40 YA = YB:YB = YA + 5:KA =  - 1:KB = 89
45  DEF  FN F(Y) = KA * Y + KB
50  GOSUB 500: REM P2
60 YA = YB:YB = YA + 5:KA =  - .4:KB = 71
65  GOSUB 500: REM P3
70 YA = YB:YB = YA + 5:KA = 0:KB = 57
75  GOSUB 500: REM P4
80 YA = YB:YB = YA + 40:KA = 57:KB = PI / 160:KC = 40
85  DEF  FN F(Y) = KA *  COS (KB * (Y - KC))
90  GOSUB 500: REM P5
100 YA = YB:YB = YA + 20:KA =  FN F(YA):KB = 5:KC = 1 / 10
105 KA = KA - KB
110  DEF  FN F(Y) = KA + KB *  EXP ( - KC * (Y - YA))
115  GOSUB 500: REM P6
120 YA = YB:YB = YA + 5:KA = 1:KB =  - 66
125  DEF  FN F(Y) = KA * Y + KB
130  GOSUB 500: REM P7
135 YA = YB:YB = YA + 10:KA =  - 1:KB = 143
140  GOSUB 500: REM P8
145 YA = YB:YB = YA + 3:KA = 0:KB = 28
150  GOSUB 500: REM P9
155  IF CASE = 1 THEN  HPLOT 107,YB TO 112,YB: HPLOT 168,YB TO 173,YB
160 YA = YB:YB = YA + 3:KB = 33
165  GOSUB 500: REM P10
170 YA = YB:YB = YA + 2:KA = 1:KB =  - 88
175  GOSUB 500: REM P11
180 YA = YB:YB = YA + 2:KA =  - 1:KB = 158
185  GOSUB 500: REM P12
190 YA = YB:YB = YA + 5:KA = 0:KB = 33
195  GOSUB 500: REM P13
200 YA = YB:YB = YA + 40:KA = 53:KB = 20:KC = PI / 40
205  DEF  FN F(Y) = KA - KB *  COS (KC * (Y - YA))
210  GOSUB 500: REM P14
215  IF CASE = 1 THEN  HPLOT 67,YB TO 213,YB
220  GOSUB 450
225  FOR D = 1 TO 3600: NEXT D
230  NEXT CASE
235  GOSUB 300: HGR2 : HCOLOR= 3
240 XC = 140:YC = 91:RC = 86:RI = 90:XF = 1.0833
245  FOR T = 0 TO 2 * PI STEP PI / 360:X = XF * RC *  COS (T):Y = RC *  SIN
     (T): HPLOT XC + X,YC + Y: NEXT T
250  FOR A = 1 TO 4 STEP .075: FOR V = 0 TO (3 + A) * PI STEP PI / (8 * A
      * A):CV =  COS (V):SV =  SIN (V)
255  HCOLOR= 3:XI = XC + XF * RI * CV:YI = YC + RI * SV: HPLOT XI,YI
260 P = A * V
265 XP = XC + XF * P * CV:YP = YC + P * SV
270  HPLOT XP,YP
275  HCOLOR= 0: HPLOT XI,YI
280  NEXT V: NEXT A
285 B$ =  CHR$ (7):B$ = B$ + B$:B$ = B$ + B$: PRINT B$,B$,B$,B$
290  GET C$
295  TEXT : END
300  TEXT : HOME : VTAB 12
305  ON CASE GOTO 310,320,330,340
310  HTAB 14: PRINT "VASE OUTLINE"
315  GOTO 350
320  HTAB 15: PRINT "LYIN' VASE": PRINT
325  HTAB 11: PRINT "(OPPOSED PROFILES)": GOTO 350
```

(Continued)

Listing 1 *(Continued)*

```
330  HTAB 16: PRINT "VASELINE"
335  PRINT : HTAB 15: PRINT "(BOTTLE OF)": GOTO 350
340  HTAB 10: PRINT "PURE PETROLEUM JELLY"
345  PRINT : HTAB 9: PRINT "(SPIRAL OF ARCHIMEDES)"
350  FOR D = 1 TO 3000: NEXT D
360  RETURN
400  ON CASE GOTO 435,405,425
405  HCOLOR= 2
410  FOR Y = 0 TO 9: HPLOT 0,Y TO 279,Y: NEXT
415  RETURN
425  HCOLOR= 1
430  FOR Y = 0 TO 9: HPLOT 67,Y TO 213,Y: NEXT
435  RETURN
450  ON CASE GOTO 490,455,460
455  HCOLOR= 2: GOTO 470
460  HCOLOR= 1
470  FOR Y = YB + 1 TO 191
475  HPLOT 0,Y TO 279,Y
480  NEXT
490  RETURN
500  FOR Y = YA TO YB
505 XF =  FN F(Y)
510 XL = XC - XF:XR = XC + XF
515  ON CASE GOTO 520,525,545
520  HPLOT XL,Y: HPLOT XR,Y: GOTO 550
525  HPLOT 0,Y TO XL,Y: HPLOT XR,Y TO 279,Y
530  IF XL <  > 2 *  INT (XL / 2) THEN XL = XL + 1
535  IF XR <  > 2 *  INT (XR / 2) THEN XR = XR - 1
540  HCOLOR= 2: HPLOT XL,Y TO XR,Y: HCOLOR= 3: GOTO 550
545  HPLOT XL,Y TO XR,Y
550  NEXT Y
555  RETURN
```


Shorthand for Cursor Control

Kerry Lourash
1220 North Dennis
Decatur, Illinois 62522

From the first moment I saw Henk Wever's program for a BASIC command "shorthand" (24:25, or *Best of Micro*, Vol. 3), I wanted to incorporate it into my cursor control program (MICRO 36:75). Wever's program allows C1P owners to print a BASIC command, such as GOSUB or RIGHT$, by hitting only two keys.

Here's an adaptation with a small improvement that adds a "(" after the string commands. I reduced the number of commands from 68 to 20 (see figure 1), since I don't see much advantage in typing (and remembering!) two keys for one-, two-, or three-letter commands.

Enter the shorthand routine by hitting the ESC key. The cursor will change from a halftone to a white square, indicating that a command key should be input. Hit the desired command key and the command will be printed on the screen. If you should accidentally hit a key that doesn't correspond to a command, the routine waits for another key.

All addresses in the table should be filled with zeroes except for the twenty command addresses listed in figure 1. If you like, you can restore Wever's original table or make your own.

Changes to the cursor control program are minimal. The ESC command of the CC is changed to "CTRL W." The PATCH option in the CC input routine is changed so that it jumps to the shorthand routine.

***Figure 1:* Shorthand Commands**

Address	Contents	Command	Key
$0282	(4E)	NEXT	N
$0283	(44)	DATA	D
$0284	(49)	INPUT	I
$0286	(52)	READ	R
$0288	(47)	GOTO	G
$028C	(3E)	GOSUB	>
$028D	(3C)	RETURN	<
$0293	(20)	LOAD	"space"
$0294	(53)	SAVE	S
$0296	(3A)	POKE	:
$0297	(3F)	PRINT	?
$0299	(4C)	LIST	L
$029C	(54)	TAB(	T
$02A0	(2D)	THEN	—
$02BB	(50)	PEEK(	P
$02BD	(2F)	STR$(	/
$02C0	(43)	CHR$(	C
$02C1	(39)	RIGHT$(	)
$02C2	(38)	LEFT$(	(
$02C3	(4D)	MID$(	M

The shift key is not used for GOSUB, RETURN, RIGHT$(, and LEFT$(. $0280-02C3 should contain zero, except for the addresses above.

Change the contents of these addresses:

$1E10 (12) to 22
$1E11 (1E) to 02
$1E5D (1B) to 17

Since I'm a thrifty sort of person, I couldn't abide the empty space at the end of page two. I managed to squeeze another routine in the space from $02C4-$02FF. This program, the CC Lister, LISTs 23 screen lines of BASIC code at a time. At the end of each set of 23 lines, you can choose to see another 23 lines by hitting the space bar. Any other key returns you to the immediate mode.

To guard against the Lister butting in when you save a program, the program checks the SAVE flag before going into action. Lister can also be defeated by typing a command before the LIST command, such as "PRINT: LIST." The contents of two addresses must be changed in the cursor control program:

$1EF0 (A8) to C4
$1EF1 (1F) to 02

If you already have a favorite routine at $0222, you could relocate these routines to the top of RAM and put the cursor control program just below them. The CC setup routine will keep them from being erased.

```
                        ;******************
                        ;*                *
                        ;* CC SHORTHAND   *
                        ;* BY K. LOURASH  *
                        ;*                *
                        ;******************
                        ;
                        TABL   EQU $280
                        PATCH  EQU $1EOF
                        ;
                               ORG $222
                        ;
                               OBJ $800
                        ;
0222 C91B                      CMP #$1B      ;SHORTHAND COMMAND?
0224 D049                      BNE RTN       ;NO, RETURN
0226 200300                    JSR $0003     ;PRINT WHITE SQUARE
0229 2000FD             GET    JSR $FD00     ;GET COMMAND KEY
022C A243                      LDX #$43      ;# OF BYTES IN TABLE-1
022E DD8002             LOOK   CMP TABL,X    ;LOOK FOR A MATCH
0231 F005                      BEQ GOT       ;FOUND IT?
0233 CA                        DEX           ;NEXT BYTE IN TABLE
0234 10F8                      BPL LOOK      ;TRY AGAIN
0236 30F1                      BMI GET       ;NO MATCH--GET ANOTHER KEY
0238 A0FF               GOT    LDY #$FF      ;SEARCH COMMAND TABLE
023A 8691                      STX $91       ;SAVE KEY INDEX
023C E8                        INX
023D CA                 G1     DEX           ;NEXT COMMAND
023E F008                      BEQ GETX      ;FOUND COMMAND?
0240 C8                 G2     INY           ;NEXT CHAR. OF COMMAND
0241 B984A0                    LDA $A084,Y   ;GET CHAR.
0244 10FA                      BPL G2        ;NOT END OF COMMAND?
0246 30F5                      BMI G1        ;END OF COMMAND
0248 68                 GETX   PLA           ;PULL Y REG.
0249 8592                      STA $92       ;SAVE IT IN $92
024B 68                        PLA           ;PULL X REG.
024C AA                        TAX
024D C8                 OUT    INY           ;PRINT THE COMMAND
024E B984A0                    LDA $A084,Y   ;GET CHAR.
0251 3005                      BMI HYBIT     ;END OF COMMAND?
0253 207202                    JSR PRIN      ;PRINT CHAR.
0256 D0F5                      BNE OUT       ;BRANCH ALWAYS
0258 297F               HYBIT  AND #$7F      ;ZERO MSB
025A 207202                    JSR PRIN      ;PRINT LAST CHAR.
025D A92C                      LDA #$2C      ;ADD "(" ?
025F C591                      CMP $91
0261 1005                      BPL DONE      ;NO, EXIT
0263 A928                      LDA #$28      ;PRINT "("
0265 207202                    JSR PRIN
0268 8A                 DONE   TXA           ;RESTORE X AND Y REG.
0269 48                        PHA
026A A592                      LDA $92
026C 48                        PHA
026D A901                      LDA #$01
026F 4C121E             RTN    JMP PATCH+3   ;EXIT ROUTINE
0272 E047               PRIN   CPX #$47      ;INPUT BUFFER FULL?
0274 B004                      BCS PO+1      ;YES, PRINT BEL CHAR.
0276 9513                      STA $13,X     ;STORE IN BUFFER
0278 E8                        INX
0279 2CA907             PO     BIT $07A9     ;LOAD BEL CHAR.
027C 6C1A02                    JMP ($021A)   ;PRINT CHAR. & RETURN
```

```
                 ;*********************
                 ;*                   *
                 ;* CC LISTER ROUTINE *
                 ;*                   *
                 ;*   BY K. LOURASH   *
                 ;*                   *
                 ;*********************

                 ;
                 RTN    EQU $1FA8
                 XIT    EQU $1FEF
                 ;
                        ORG $2C4
                 ;

                 ;
02C4 AE0502             LDX $205     ;CHECK SAVE FLAG
02C7 D01D               BNE CONT
02C9 A613               LDX $13      ;IS LIST STARTING?
02CB E099               CPX #$99
02CD D006               BNE FLAG     ;NO, CHECK LIST FLAG
02CF A017               LDY #$17     ;SET LINE COUNTER
02D1 8414               STY $14
02D3 E613               INC $13      ;TURN ON LIST FLAG
02D5 E09A        FLAG   CPX #$9A     ;IS LIST FLAG ON?
02D7 D00D               BNE CONT     ;NO, TO REG. OUTPUT
02D9 C614               DEC $14      ;DECREMENT COUNTER
02DB D009               BNE CONT     ;IS COUNTER ZERO?
02DD 2000FD             JSR $FD00    ;GET CHAR. FROM KYBD.
02E0 C920               CMP #$20     ;IS IT A SPACE?
02E2 D005               BNE STOP     ;IF NOT, STOP
02E4 C613               DEC $13      ;TURN OFF FLAG
02E6 4CA81F      CONT   JMP RTN      ;TO REGULAR OUTPUT
02E9 4CEF1F      STOP   JMP XIT      ;TO IMMEDIATE MODE
```

MICRO

Applesoft Variable Lister

The ability to dump the values of all variables can be immensely helpful in Applesoft program development. The Applesoft Variable Lister provides this ability and can be used with any program, located anywhere in memory.

Richard Albright
Sienna Software
25 Marion Road
Watertown, Massachusetts 02172

Scott Schram, in his "Applesoft Variable Dump" article (MICRO 36:23), presented a machine language program for printing the current values of all Applesoft variables in use. Such a program is immensely useful in developing and debugging Applesoft programs because it permits the programmer to easily display the values of all simple variables at any time. Also, by providing a list of used variable names, it helps prevent the accidental duplication of names, which is an easy mistake to make with Applesoft's two-character names.

I used Schram's routine successfully on a number of Applesoft programs, but I also discovered a number of programs on which the routine did not work at all or did not work well. The primary problem lies in its need to be loaded at $4000. This location is the start of Applesoft's Hi-Res page two. Therefore any program using Hi-Res page two cannot use Schram's routine. Moreover, any program loaded above Hi-Res page one (virtually a requirement for any large program using Hi-Res graphics) will probably spill over the $4000 boundary.

I also attempted to use this routine on a program having nearly 100 simple variables and discovered how difficult it is to find the value of a single variable in an unsorted list of that length. The inability to list the names of array variables was also troublesome at times.

After trying to modify Schram's routine, I decided that a different approach was needed. My approach involves three routines — an Applesoft subroutine and two machine language routines. I will refer to these three routines collectively as the "Applesoft Variable Lister" (or simply "Lister").

Installing the Lister

The Applesoft Variable Lister may be attached to any Applesoft program by simply merging its Applesoft subroutine with the main program. This can be accomplished using the standard Apple RENUMBER program or the like. Any unused space in which the 71 lines will fit without affecting the normal operation of the program will do, but the end of the program is the recommended location.

Once installed within the program, the Lister can be invoked like any Applesoft subroutine; that is, by means of a GOSUB n statement where n is the number of the first line of the subroutine within the program. This GOSUB can be issued by the main program or from the keyboard.

The Lister will operate under both ROM and RAM Applesoft, but requires the use of a disk drive. The disk drive last accessed before the Lister was invoked must contain a diskette on which the Lister's two machine language routines are stored under the names SHELL-METZNER SORT and APPLESOFT VARIABLE LISTER OBJ. In addition, one file buffer must be available.

Using the Lister

The output from the Lister will appear on both a printer and the screen if the printer is open at the time the Lister is invoked. Otherwise, the output goes to the screen only. The output format for the printer is slightly different from the screen format.

Figure 1 is an example of the printed output format. User responses to prompts have been underlined. When the Lister is invoked, it first queries you for

ALPHA SORT, MEMORY SORT

OR QUIT?

with the double-underlined letters appearing in inverse on the screen. A 'Q' response at this point simply terminates the Lister with no further ado. An 'A' response results in an alphabetical listing of variables while an 'M' response will cause variables to be listed in the order stored. After either an 'A' or an 'M' response, the disk drive will activate briefly while a temporary file is created (more on this later).

Next, the Lister asks if you would like to display

VALUES OR LOCATIONS?

A 'V' response will give you the current value for each simple variable (as shown in figure 1); an 'L' response produces a display of locations at which the values are stored in memory.

At this point the disk drive will again activate while the APPLESOFT VARIABLE LISTER OBJ and (if ALPHA SORT has been selected) the SHELL-METZNER SORT files are read and another temporary file is created. If sorting is performed, a

SORTING VARIABLE NAMES . . .

MICRObits

(Continued on page 90)

***Figure 1:* Example of Printed Output**

APPLESOFT VARIABLE LISTER

ALPHA SORT, MEMORY SORT OR QUIT? A

VALUES OR LOCATIONS? V

SORTING VARIABLE NAMES...

Simple Variables; Alpha Order

Var	Value	Var	Value
A	0	LB$	[
A $		ML%	3
B1$	3 LETTERS	MQ	99
B2$	0	MR	99
BS$		NL%	12
CA$		NQ%	9
CL$	CLOSE	NR	12
CR$			
		NR%	0
D	0	NS%	10
D $		O0$	OPEN SURVEY C
EQ	1	O1$	OPEN SURVEY T
ER%	1	O2$	OPEN INTERVIE
F1$	TEST1	OP$	OPEN SURVEY T
FQ%	9	PP	0
I	96	Q	1
J	0	QQ	1
K	0	R	0
L	0	R0$	READ SURVEY C
L1	9	R1$	READ SURVEY T
L2	1	RE$	READ SURVEY C
RR	1		
RT%	4		
SS	0		
T $			
UN	2048		
V	5		
W2$	WRITE INTERVI		
XR%	I		
ZZ	I		

Array Variables; Alpha Order

Var	Hex	Dec	Var	Hex	Dec
CT	$2DB2	11698			
DT$	$33F5	13301			
QC	$2F99	12185			
R $	$3194	12692			

message is displayed while the names are being sorted. Usually the sorting process takes only a few seconds.

After a slight pause, the first page of variables will be displayed (and printed if the printer is on). A two-column format is used for all combinations of display options. Numeric values are displayed to full precision, but strings longer than 14 characters are truncated. Forty variables appear on a full page. The message

HIT SPACE BAR TO CONTINUE;
'ESC' TO QUIT

appears on the screen (*not* on the printer) after each page. Pressing the ESC key results in the termination of the Lister (after some more disk activity). Pressing the space bar, on the other hand, causes the next page of simple variables to be displayed. If all simple variables have been displayed, the first page of array variables is produced. Notice that array variable values *cannot* be displayed; only the location of the start of each array is provided — even if VALUES is the selected display mode.

Following the last array page, the Lister is terminated by pressing either the space bar or the ESC key. At this

point the disk drive will again briefly activate. If the Lister was invoked from the keyboard, a

RETURN WITHOUT GOSUB

error message will be encountered and can be ignored. If invoked from the main program, execution continues normally with the statement following the GOSUB.

The Source Code

The Applesoft Variable Lister consists of an Applesoft subroutine (listing 1), a machine language setup routine (listing 2), and a machine language sort routine (listing 3). The Applesoft subroutine can be entered and SAVEd under an arbitrary name. The machine language routines may be entered into memory either directly using the monitor or indirectly using an assembler, then BSAVEd under the names APPLESOFT VARIABLE LISTER OBJ (for the setup routine) and SHELL-METZNER SORT (for the sort routine).

Technical Notes

The Lister's Applesoft subroutine occupies about 3500 bytes of memory. In addition, execution of the Lister requires a certain amount of free space: five bytes per variable if the ALPHA SORT option is chosen and ten bytes per variable if the MEMORY SORT option is selected. The Lister does *not* verify that this space is available. If insufficient space exists, the result is unpredictable.

If the addition of the Lister to a program using Hi-Res graphics causes the program to overflow into the Hi-Res memory area, then the merged program should be saved and reloaded above the Hi-Res memory. If only Hi-Res page one is used, this move is accomplished by executing the following POKEs between the SAVE and the LOAD:

POKE 103,1:POKE 104,64:POKE 16384,0

To move the program above Hi-Res page two, use the following POKEs:

POKE 103,1:POKE 104,96:POKE 24576,0

The Lister's Applesoft subroutine itself uses three simple variables (ZZ, ZZ% and ZZ$) and one array variable (ZZ). These variable names should be

Listing 1: **Applesoft Variable Lister**

```
10  FOR ZZ = 32 TO 35: POKE 715 + ZZ, PEEK (ZZ): NEXT ZZ
20  POKE 32,0: POKE 33,40: POKE 34,0: POKE 35,24: TEXT : NORMAL
30  PRINT : INVERSE : PRINT  SPC( 7);"APPLESOFT VARIABLE LISTER": SPC( 8)
     : NORMAL
40  FOR ZZ = 0 TO 9: POKE 752 + ZZ,48 + ZZ: NEXT ZZ: FOR ZZ = 10 TO 15: POKE
     752 + ZZ,55 + ZZ: NEXT ZZ
50  PRINT : INVERSE : PRINT "A";: NORMAL : PRINT "LPHA SORT, ";: INVERSE
     : PRINT "M";: NORMAL : PRINT "EMORY SORT OR ";: INVERSE : PRINT "Q";
     : NORMAL : PRINT "UIT? ";
60 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 60
70  POKE  - 16368,0: PRINT  CHR$ (ZZ): IF ZZ <  > 193 AND ZZ <  > 205 AND
     ZZ <  > 209 THEN  PRINT  CHR$ (7): GOTO 50
80  IF ZZ = 209 THEN 700
90 ZZ = ZZ - 192: IF ZZ > 1 THEN ZZ = 2
100  POKE 250,ZZ: INVERSE : PRINT "V";: NORMAL : PRINT "ALUES OR ";: INVERSE
     : PRINT "L";: NORMAL : PRINT "OCATIONS? ";
110 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 110
120  POKE  - 16368,0: PRINT  CHR$ (ZZ): IF ZZ <  > 204 AND ZZ <  > 214 THEN
      PRINT  CHR$ (7): GOTO 120
130 ZZ = ZZ - 204: IF ZZ > 0 THEN ZZ = 2
140 ZZ = ZZ +  PEEK (250)
150  PRINT  CHR$ (4);"BSAVE PAGE 3 SAVE,A$300,L$100": PRINT  CHR$ (4);"BS
     AVE PAGE 0 SAVE,A$C0,L$40"
160  PRINT  CHR$ (4);"BLOAD APPLESOFT VARIABLE LISTER OBJ": PRINT  CHR$ (
     4)
170  POKE 250,ZZ:ZZ =  FRE (0): CALL 768
180  POKE 251, PEEK (111): POKE 252, PEEK (112): IF  PEEK (250) = 2 OR  PEEK
     (250) = 4 THEN 260
190  PRINT  CHR$ (4);"BSAVE PAGE 0 SAVE2,A$C0,L$40": PRINT  CHR$ (4)
200  PRINT  CHR$ (4);"BLOAD SHELL-METZNER SORT": PRINT  CHR$ (4)
210  PRINT : PRINT "SORTING VARIABLE NAMES . . .": PRINT
220  POKE 25,5: POKE 26,0: POKE 27,3
230 ZZ =  PEEK (251) + 256 *  PEEK (252) + 5 *  PEEK (254): POKE 28, PEEK
     (253): POKE 29,0: POKE 31, INT (ZZ / 256): POKE 30,ZZ - 256 *  PEEK
     (31):ZZ =  PEEK (254): CALL 768
240  POKE 28,ZZ: POKE 29,0:ZZ =  PEEK (30) + 256 *  PEEK (31) - 5 * ZZ: POKE
     31, INT (ZZ / 256): POKE 30,ZZ - 256 *  PEEK (31): CALL 768
250  PRINT  CHR$ (4);"BLOAD PAGE 0 SAVE2": PRINT  CHR$ (4);"DELETE PAGE 0
      SAVE2": PRINT  CHR$ (4)
260  HOME : INVERSE : PRINT  SPC( 5);"SIMPLE VARIABLES; ";: IF  PEEK (250
     ) = 1 OR  PEEK (250) = 3 THEN  PRINT "ALPHA ORDER"; SPC( 6);
270  IF  PEEK (250) = 2 OR  PEEK (250) = 4 THEN  PRINT "MEMORY ORDER"; SPC(
     5);
280  PRINT : NORMAL : IF  PEEK (253) = 0 THEN  PRINT : PRINT "NO SIMPLE V
     ARIABLES": GOSUB 400: GOTO 320
290 ZZ(0) =  PEEK (253):ZZ(1) =  PEEK (251) + 256 *  PEEK (252) + 5 * ( PEEK
     (253) +  PEEK (254))
300  IF  PEEK (250) > 2 THEN ZZ = ZZ: POKE 25, PEEK (131): POKE 26, PEEK
     (132):ZZ$ = ZZ$: POKE 27, PEEK (131): POKE 28, PEEK (132):ZZ% = ZZ%:
      POKE 29, PEEK (131): POKE 30, PEEK (132)
310  GOSUB 450
320  IF  PEEK (250) > 2 THEN  POKE 250, PEEK (250) - 2
330  HOME : INVERSE : PRINT  SPC( 6);"ARRAY VARIABLES; ";: IF  PEEK (250)
      = 1 THEN  PRINT "ALPHA ORDER"; SPC( 6);
340  IF  PEEK (250) = 2 THEN  PRINT "MEMORY ORDER"; SPC( 5);
```

```
350  PRINT : NORMAL : IF  PEEK (254) = 0 THEN  PRINT : PRINT "NO ARRAY VA
     RIABLES": GOSUB 400: GOTO 370
360 ZZ(0) =  PEEK (254):ZZ(1) =  PEEK (251) + 256 *  PEEK (252) + 5 *  PEEK
     (254): GOSUB 450
370  GOTO 660
380  VTAB 2: PRINT "VAR HEX   DEC      * VAR HEX   DEC": PRINT "--- -----
      -----    * --- ----- -----": RETURN
390  VTAB 2: PRINT "VAR VALUE               * VAR VALUE": PRINT "--- ---------
     ----- * --- --------------": RETURN
400 ZZ$ = "HIT" +  CHR$ (96) + "SPACE" +  CHR$ (96) + "BAR" +  CHR$ (96) +
     "TO" +  CHR$ (96) + "CONTINUE" +  CHR$ (123) +  CHR$ (96) +  CHR$ (1
     03) + "ESC" +  CHR$ (103) +  CHR$ (96) + "TO" +  CHR$ (96) + "QUIT"
410  FOR ZZ = 1 TO  LEN (ZZ$): POKE ZZ + 1999, ASC ( MID$ (ZZ$,ZZ,1)) - 6
     4: NEXT ZZ
420 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 420
430  POKE  - 16368,0: IF ZZ <  > 155 THEN  PRINT : PRINT : RETURN
440  POP : POP : GOTO 660
450  REM  PRINT VARIABLE NAMES & LOCATIONS
460 ZZ(10) =  INT (( PEEK (250) + 1) / 2): ON ZZ(10) GOSUB 380,390: POKE
     34,3
470 ZZ(3) = 0:ZZ(1) = ZZ(1) - 5
480 ZZ(2) = ZZ(3) + 1: IF ZZ(2) > ZZ(0) THEN  POKE 34,0: RETURN
490 ZZ(3) = ZZ(2) + 19: IF ZZ(3) > ZZ(0) THEN ZZ(3) = ZZ(0)
500 ZZ(6) = ZZ(2) - 1
510 ZZ(6) = ZZ(6) + 1: IF ZZ(6) > ZZ(3) THEN ZZ(1) = ZZ(1) - 100:ZZ(3) =
     ZZ(3) + 20: GOSUB 400: HOME : GOTO 480
520  VTAB ZZ(6) - ZZ(2) + 4:ZZ(8) = ZZ(1): GOSUB 540: PRINT  SPC( 19 -  POS
     (0));"* ";: IF ZZ(6) + 20 <  = ZZ(0) THEN ZZ(8) = ZZ(1) - 100: GOSUB
     540
530  PRINT :ZZ(1) = ZZ(1) - 5: GOTO 510
540  PRINT  CHR$ ( PEEK (ZZ(8))); CHR$ ( PEEK (ZZ(8) + 1)); CHR$ ( PEEK (
     ZZ(8) + 2));" ";: IF ZZ(10) = 2 THEN 600
550  PRINT "$";:ZZ(5) =  PEEK (ZZ(8) + 4):ZZ(4) =  PEEK (ZZ(8) + 3):ZZ(7)
      =  INT (ZZ(5) / 16): PRINT  CHR$ ( PEEK (752 + ZZ(7))); CHR$ ( PEEK
     (752 + ZZ(5) - 16 * ZZ(7)));
560 ZZ(7) =  INT (ZZ(4) / 16): PRINT  CHR$ ( PEEK (752 + ZZ(7))); CHR$ ( PEEK
     (752 + ZZ(4) - 16 * ZZ(7)));
570 ZZ$ =  STR$ (256 * ZZ(5) + ZZ(4))
580  PRINT  SPC( 6 -  LEN (ZZ$));ZZ$;
590  RETURN
600 ZZ(9) =  PEEK (ZZ(8) + 3) + 256 *  PEEK (ZZ(8) + 4):ZZ =  PEEK (ZZ(8)
      + 2) - 31: IF ZZ > 1 THEN ZZ = ZZ - 3
610  ON ZZ GOTO 620,640,650
620 ZZ(7) =  PEEK (25) + 256 *  PEEK (26) - 2: POKE ZZ(7) + 2, PEEK (ZZ(9
     ) + 2): POKE ZZ(7) + 3, PEEK (ZZ(9) + 3): POKE ZZ(7) + 4, PEEK (ZZ(9
     ) + 4): POKE ZZ(7) + 5, PEEK (ZZ(9) + 5)
630  POKE ZZ(7) + 6, PEEK (ZZ(9) + 6): PRINT ZZ;: RETURN
640 ZZ(7) =  PEEK (27) + 256 *  PEEK (28) - 2: FOR ZZ = 2 TO 4: POKE ZZ(7
     ) + ZZ, PEEK (ZZ(9) + ZZ): NEXT ZZ: PRINT  LEFT$ (ZZ$,14);: RETURN
650 ZZ(7) =  PEEK (29) + 256 *  PEEK (30) - 2: FOR ZZ = 2 TO 3: POKE ZZ(7
     ) + ZZ, PEEK (ZZ(9) + ZZ): NEXT ZZ: PRINT ZZ%;: RETURN
660  IF ZZ = 209 THEN 700
670  HOME : PRINT : PRINT  CHR$ (4);"BLOAD PAGE 0 SAVE": PRINT  CHR$ (4);
     "DELETE PAGE 0 SAVE": PRINT  CHR$ (4)
680  PRINT  CHR$ (4);"BLOAD PAGE 3 SAVE"
690  PRINT  CHR$ (4);"DELETE PAGE 3 SAVE": PRINT  CHR$ (4)
700  FOR ZZ = 32 TO 35: POKE ZZ, PEEK (715 + ZZ): NEXT ZZ
710  HOME : RETURN
```

avoided in the main program: if they appear in the main program, execution of the Lister subroutine will reset their values.

Both the SHELL-METZNER SORT and APPLESOFT VARIABLE LISTER OBJ routines use page three of memory. However, the contents of page three at the time the Lister is invoked are saved on diskette in a temporary file named PAGE 3 SAVE. The original page three is restored as part of the Lister termination processing.

Both machine language routines make extensive use of page zero, but, again, a temporary file (PAGE 0 SAVE) is used to save the initial values and they are restored when the Lister finishes. However, only part of page zero is restored, leaving some page zero values altered after running the Lister. Specifically, locations 24 to 31 ($18 to $1F) are altered. These locations are not normally used by an Applesoft program.

A third temporary file (PAGE 0 SAVE2) is used if ALPHA SORT is selected. It is used to restore page zero values after the sorting has been completed. All temporary files are deleted by the Lister if it terminates normally. Both the SHELL-METZNER SORT and the APPLESOFT VARIABLE LISTER OBJ routines are fully relocatable.

The sorting routine uses the Shell-Metzner algorithm and is designed to sort fixed-length records so that the one with the lowest key value appears highest in the memory. Up to 32,767 records occupying contiguous locations may be sorted with this routine, space permitting. Each record may be up to 255 bytes in length and must have a sort key field that may be as short as one byte or as long as the entire record. The key is evaluated as an unsigned binary integer field and the sorting is performed on that basis.

The sort routine uses memory locations 25 to 31 ($19 to $1F) as an input argument list, interpreted as follows:

25	($19):	record length
26	($1A):	key offset (i.e., record characters preceding the key)
27	($1B):	key length
28-29	($1C-$1D):	number of records
30-31	($1E-$1F):	pointer to 1st byte of 1st record.

The last two items are two-byte binary integers, presented in the usual low

Listing 2: The APPLESOFT VARIABLE LISTER OBJ Routine

```
                 ****************************************
                 *                                      *
                 *      APPLESOFT VARIABLE LISTER OBJ   *
                 *      ============================    *
                 *                                      *
                 *            SEPTEMBER 30, 1981        *
                 *                                      *
                 * THIS ROUTINE CREATES AN N-BY-5 ARRAY *
                 * OF APPLESOFT VARIABLE INFORMATION AT *
                 * THE BOTTOM OF THE STRING STORAGE AREA.*
                 * EACH 5-BYTE 'RECORD' CONTAINS A      *
                 * 3-BYTE NAME AND A 2-BYTE POINTER TO  *
                 * ITS LOCATION IN MEMORY.              *
                 *                                      *
                 ****************************************
                 *
                 VNAME   EQU  $A5          ;CURRENT VARIABLE NAME
                 VLOC    EQU  $A8          ;CURRENT VARIABLE LOCATION
                 VTYPE   EQU  $AA          ;VARIABLE TYPE(0=SIMPLE;1=ARRAY)
                 NSIMPL  EQU  $FD          ;COUNT OF SIMPLE VARIABLES
                 NARRAY  EQU  $FE          ;COUNT OF ARRAY VARIABLES
                 *

0300:                    ORG  $300
0300:            *
0300:A5 69               LDA  $69          ;INITIALIZE VARIABLE POINTER TO
0302:85 A8               STA  VLOC         ;START OF SIMPLE VARIABLE
0304:A5 6A               LDA  $6A          ;SPACE
0306:85 A9               STA  VLOC+1
0308:A9 00               LDA  #$00         ;INITIALIZE VARIABLE COUNTERS
030A:85 FD               STA  NSIMPL       ;TO ZERO
030C:85 FE               STA  NARRAY
030E:85 AA               STA  VTYPE        ;START WITH SIMPLE VARIABLES
0310:A5 AA       TOP     LDA  VTYPE        ;TOP OF MAIN LOOP
0312:18                  CLC
0313:65 AA               ADC  VTYPE        ;SET X TO 2 TIMES THE
0315:AA                  TAX               ;VARIABLE INDEX
0316:A5 A9               LDA  VLOC+1       ;IF CURRENT VARIABLE IS NOT
0318:D5 6C               CMP  $6C,X        ;BEYOND THE END OF THE
031A:90 11               BCC  STRTVP       ;STORAGE SPACE FOR THE
031C:D0 06               BNE  INCVT        ;CURRENT VARIABLE TYPE,
031E:A5 A8               LDA  VLOC         ;THEN GO ON TO VARIABLE
0320:D5 6B               CMP  $6B,X        ;PROCESSING
0322:90 09               BCC  STRTVP
0324:E6 AA       INCVT   INC  VTYPE        ;INCREMENT VARIABLE TYPE
0326:A4 AA               LDY  VTYPE
0328:C0 02               CPY  #$02
032A:D0 E4               BNE  TOP          ;GO BACK TO THE TOP IF INDEX<>2
032C:60                  RTS               ;QUIT IF INDEX=2
032D:A6 AA       STRTVP  LDX  VTYPE        ;START OF VARIABLE PROCESSING
032F:F6 FD               INC  NSIMPL,X     ;INCREMENT VARIABLE COUNT
0331:A2 00               LDX  #$00         ;BLANK OUT CURRENT VARIABLE
0333:A9 20               LDA  #$20         ;NAME
0335:95 A5       BLNKVN  STA  VNAME,X
0337:E8                  INX
0338:E0 03               CPX  #$03
033A:D0 F9               BNE  BLNKVN
033C:A0 00               LDY  #$00         ;IF BIT 7 IS OFF, THEN
033E:B1 A8               LDA  (VLOC),Y     ;SKIP INTEGER PROCESSING
0340:C9 7F               CMP  #$7F
0342:90 18               BCC  SAVE1
0344:A2 25               LDX  #$25         ;ATTACH '%' TO NAME
0346:86 A7               STX  VNAME+2
0348:29 7F               AND  #$7F         ;SAVE 1ST CHARACTER
034A:85 A5               STA  VNAME
034C:C8                  INY               ;STRIP BIT 7 FROM 2ND CHARACTER
034D:B1 A8               LDA  (VLOC),Y     ;AND SAVE IF NOT $00
034F:29 7F               AND  #$7F
0351:C9 00               CMP  #$00
0353:F0 1C               BEQ  LOWER
0355:85 A6               STA  VNAME+1
0357:18                  CLC               ;SKIP STRING PROCESSING
0358:90 17               BCC  LOWER
035A:90 B4       RELAY   BCC  TOP          ;RELAY RETURN TO TOP
035C:85 A5       SAVE1   STA  VNAME        ;SAVE 1ST CHARACTER
035E:C8                  INY               ;GET 2ND
035F:B1 A8               LDA  (VLOC),Y
0361:C9 7F               CMP  #$7F         ;IF BIT 7 IS OFF, THEN
0363:90 06               BCC  SAVE2        ;SKIP STRING PROCESSING
0365:A2 24               LDX  #$24         ;ATTACH '$' TO NAME
0367:86 A7               STX  VNAME+2
0369:29 7F               AND  #$7F         ;STRIP BIT 7
036B:C9 00       SAVE2   CMP  #$00         ;SAVE 2ND CHARACTER IF NOT ZERO
```

byte/high byte format. The sorting routine does not alter the values placed in any of these locations, nor does it verify their consistency.

Although the sort routine can handle thousands of records, the setup routine can handle a maximum of 255 variables of any types (simple or array). If more than 255 simple or array variables exist, the operation of the Lister is unpredictable.

Strings containing one or more carriage return characters (ASCII 13) cause formatting problems on both the screen and the printer. If the value appears in the left column on the screen, then one variable may be omitted from the right column. On the printer, one or more blank lines may be introduced. This problem is exemplified in figure 1: the CR$ string consists of a single carriage return character, resulting in the unexpected gap between the CR$ and D variables in the left column and the NR and NR% variables in the right column.

Conclusion

In spite of its minor restrictions, I have found the Applesoft Variable Lister to be a valuable programming aid. If you have any comments or suggestions for improving the Lister, I would like to hear from you. Write me at the address given at the beginning of this article.

MICRObits *(continued)*

Used Computer Exchange

Save time, money and mistakes. List as Buyer/Seller — get pricing advice with names and phone numbers of those who meet your criteria on first call. Apples, PETs, Atari, OSI, printers, CRTs, etc. 600+ listed. Pay only for results. Call:

(703) 471-0305 or
(800) 336-3393
Used Computer Exchange
11484 Washington Plaza West
Reston, VA 22090

OSI Superboard/CIP Expension Board

Adds 8K 2114 RAM and 4 EPROM sockets (2716 or 2732). All link addressable anywhere in memory, all lines buffered, plugs into expansion socket. Bare PCB $39.95. Built (no RAM) $99.95. Payment: check or Mastercard. Includes airmail return.

Northern Micro
29 Moorcroft Park
New Mill
Huddersfield, England

OSI Toolkit EPROM

2716 EPROM. Address 9800-9FFF. Adds 16 BASIC functions: RENUMBER, HEX/DEC, DEC/HEX, CONVERSION, CONTROLLED LIST, VARIABLE LIST, TRACE, VIEW, SEARCH, etc. $39.95. Also: Assembler (3 × 2716) $49.95. Exmon EPROM $19.95. Payment: check or Mastercard. Includes airmail return.

Northern Micro
29 Moorcroft Park
New Mill
Huddersfield, England

Printer Spooling for OSI

Output directed to device #1 is automatically spooled to disk tracks. Outspool to printer at your leisure. Outspool features multiple copies and auto-paging. 5¼" dual floppy systems only. Specify RAM size and 65D rev (3.2 or 3.3). $20.

Peter Koski
144-F Delaware Avenue
Troy, New York 12180

```
036D:F0 02              BEQ  LOWER
036F:85 A6              STA  VNAME+1
0371:38        LOWER    SEC                 ;LOWER START OF STRING
0372:A5 6F              LDA  $6F            ;STORAGE AREA BY 5
0374:E9 05              SBC  #$05
0376:85 6F              STA  $6F
0378:A5 70              LDA  $70
037A:E9 00              SBC  #$00
037C:85 70              STA  $70
037E:A0 00              LDY  #$00           ;MOVE VARIABLE DESCRIPTION
0380:B9 A5 00 MOVE      LDA  VNAME,Y        ;TO STRING STORAGE
0383:91 6F              STA  ($6F),Y
0385:C8                 INY
0386:C0 05              CPY  #$05
0388:D0 F6              BNE  MOVE
038A:A5 AA              LDA  VTYPE          ;IF CURRENT VARIABLE TYPE=1
038C:C9 01              CMP  #$01           ;(I.E., AN ARRAY VARIABLE),
038E:F0 10              BEQ  INCPTR         ;SKIP SIMPLE VARIABLE
0390:18                 CLC                 ;INCREMENT CURRENT VARIABLE
0391:A5 A8              LDA  VLOC           ;LOCATION BY 7 AND GO ON
0393:69 07              ADC  #$07           ;TO THE NEXT VARIABLE
0395:85 A8              STA  VLOC
0397:A5 A9              LDA  VLOC+1
0399:69 00              ADC  #$00
039B:85 A9              STA  VLOC+1
039D:18                 CLC
039E:90 11              BCC  GETNXT
03A0:A0 02     INCPTR   LDY  #$02           ;INCREMENT CURRENT VARIABLE
03A2:18                 CLC                 ;LOCATION BY THE LENGTH
03A3:A5 A8              LDA  VLOC           ;OF THE CURRENT ARRAY
03A5:71 A8              ADC  (VLOC),Y       ;AND GO ON TO THE
03A7:AA                 TAX                 ;NEXT ARRAY VARIABLE
03A8:A5 A9              LDA  VLOC+1
03AA:C8                 INY
03AB:71 A8              ADC  (VLOC),Y
03AD:85 A9              STA  VLOC+1
03AF:86 A8              STX  VLOC
03B1:18        GETNXT   CLC                 ;GO ON TO THE NEXT
03B2:90 A6              BCC  RELAY          ;VARIABLE
```

Listing 3: **The SHELL-METZNER SORT Routine**

```
         ;**********************************************
         *           SHELL-METZNER SORT               *
         *           ==================               *
         *                                            *
         *           OCTOBER  4, 1981                 *
         *                                            *
         * THIS ROUTINE USES THE SHELL-METZNER        *
         * SORTING ALGORITHM TO SORT 'N' RECORDS OF   *
         * 'RL' BYTES EACH SO THAT FOR ANY RECORD     *
         * ALL RECORDS HAVING LOWER KEY VAUES         *
         * APPEAR ABOVE IT IN MEMORY.  EACH RECORD    *
         * MAY BE UP TO 255 BYTES IN LENGTH AND THE   *
         * SINGLE KEY FIELD MAY BE AS SHORT AS ONE    *
         * BYTE OR AS LONG AS THE ENTIRE RECORD.      *
         * THE KEY FIELD IS TREATED AS A BINARY       *
         * INTEGER.  ALL RECORDS MUST EXIST IN        *
         * CONTIGUOUS MEMORY LOCATIONS.               *
         *                                            *
         **********************************************
         *
         RL      EQU  $19        ;RECORD LENGTH
         KEYOFF  EQU  $1A        ;KEY OFFSET FROM START OF RECORD
         KEYLEN  EQU  $1B        ;KEY LENGTH
         N       EQU  $1C        ;NUMBER OF RECORDS IN $1C-$1D
         ARRAY   EQU  $1E        ;POINTER TO ARRAY IN $1E-$1F
         KEYEND  EQU  $C9        ;OFFSET OF LAST KEY BYTE
         I       EQU  $CA        ;INDEX I IN $CA-$CB
         L       EQU  $CC        ;INDEX L IN $CC-$CD
         M       EQU  $CE        ;INDEX M IN $CE-$CF
         K       EQU  $D6        ;INDEX K IN $D6-$D7
         J       EQU  $EB        ;INDEX J IN $EB-$EC
         CNT1    EQU  $FA        ;TEMPORARY COUNTERS IN $FA-$FF
         CNT2    EQU  $FC
         CNT3    EQU  $FE
         *

0300:            ORG  $300
0300:    *
0300:18          CLC             ;ESTABLISH OFFSET OF LAST
0301:A5 1A       LDA  KEYOFF     ;KEY BYTE
0303:65 1B       ADC  KEYLEN
0305:85 C9       STA  KEYEND
```

```
0307:A5 1C            LDA   N           ;INITIALIZE M TO N
0309:85 CE            STA   M
030B:A5 1D            LDA   N+1
030D:85 CF            STA   M+1
030F:18      LOOP1    CLC               ;TOP OF MAIN LOOP
0310:66 CF            ROR   M+1         ;M := M/2
0312:66 CE            ROR   M
0314:A5 CE            LDA   M           ;STOP IF M=0
0316:D0 05            BNE   MORE
0318:A5 CF            LDA   M+1
031A:D0 01            BNE   MORE
031C:60               RTS
031D:A2 00   MORE     LDX   #$00        ;K := N-M
031F:38               SEC
0320:A5 1C            LDA   N
0322:E5 CE            SBC   M
0324:85 D6            STA   K
0326:A5 1D            LDA   N+1
0328:E5 CF            SBC   M+1
032A:85 D7            STA   K+1
032C:A9 01            LDA   #$01        ;J := 1
032E:85 EB            STA   J
0330:A9 00            LDA   #$00
0332:85 EC            STA   J+1
0334:A5 EB   LOOP2    LDA   J           ;I := J
0336:85 CA            STA   I
0338:A5 EC            LDA   J+1
033A:85 CB            STA   I+1
033C:18      LOOP3    CLC               ;L := I+M
033D:A5 CA            LDA   I
033F:65 CE            ADC   M
0341:85 CC            STA   L
0343:A5 CB            LDA   I+1
0345:65 CF            ADC   M+1
0347:85 CD            STA   L+1
0349:A2 00            LDX   #$00        ;SET X REGISTER TO 0
034B:A4 19   GETLOC   LDY   RL          ;SET Y REGISTER TO RECORD LENGTH
034D:38               SEC               ;INITIALIZE CNT2 TO I-1
034E:B5 CA            LDA   I,X         ;IF X=0
0350:E9 01            SBC   #$01        ;INITIALIZE CNT3 TO L-1
0352:85 FA            STA   CNT1        ;IF X=2
0354:95 FC            STA   CNT2,X      ;AND STORE THE SAME
0356:B5 CB            LDA   I+1,X       ;VALUE IN CNT1
0358:E9 00            SBC   #$00
035A:85 FB            STA   CNT1+1
035C:95 FD            STA   CNT2+1,X
035E:88      GETOFF   DEY               ;MULTIPLY BY RECORD LENGTH TO
035F:F0 15            BEQ   GETABS      ;GET THE OFFSET DF THE
0361:18               CLC               ;(I-1)TH RECORD (IF X=0) OR THE
0362:A5 FA            LDA   CNT1        ;(L-1)TH RECORD (IF X=2) FROM
0364:75 FC            ADC   CNT2,X      ;THE START OF THE ARRAY
0366:95 FC            STA   CNT2,X
0368:A5 FB            LDA   CNT1+1
036A:75 FD            ADC   CNT2+1,X
036C:95 FD            STA   CNT2+1,X
036E:90 EE            BCC   GETOFF
0370:D0 CA   RELAY3   BNE   LOOP3       ;RELAY RETURNS
0372:90 C0   RELAY2   BCC   LOOP2
0374:D0 99   RELAY1   BNE   LOOP1
0376:18      GETABS   CLC               ;ADD LOCATION OF START
0377:A5 1E            LDA   ARRAY       ;OF ARRAY TO GET ABSOLUTE
0379:75 FC            ADC   CNT2,X      ;LOCATIDN DF (1-1)TH OR
037B:95 FC            STA   CNT2,X      ;(L-1)TH RECORD
037D:A5 IF            LDA   ARRAY+1
037F:75 FD            ADC   CNT2+I,X
0381:95 FD            STA   CNT2+1,X
0383:E8               INX               ;ADD 2 TO X REGISTER
0384:E8               INX
0385:E0 04            CPX   #$04        ;GO GET (L-1)TH RECORD
0387:D0 C2            BNE   GETLDC      ;IF X=2
0389:A4 1A            LDY   KEYOFF      ;SET Y REGISTER TO KEY OFFSET
038B:B1 FC   COMPAR   LDA   (CNT2),Y    ;COMPARE (I-1)TH AND
038D:D1 FE            CMP   (CNT3),Y    ;(L-1)TH KEY VALUES;
038F:90 09            BCC   SWITCH      ;SWITCH RECORDS IF THE
0391:D0 2F            BNE   INCJ        ;(L-1)TH KEY IS > THE
0393:C8               INY               ;(I-1)TH KEY
0394:C4 C9            CPY   KEYEND
0396:D0 F3            BNE   COMPAR
0398:F0 28            BEQ   INCJ
039A:A4 I9   SWITCH   LDY   RL
039C:88      SW1      DEY
039D:B1 FC            LDA   (CNT2),Y
039F:AA               TAX
03A0:B1 FE            LDA   (CNT3),Y
03A2:91 FC            STA   (CNT2),Y
03A4:8A               TXA
```

(Continued)

```
03A5:91 FE                 STA  (CNT3),Y
03A7:C0 00                 CPY  #$00
03A9:D0 F1                 BNE  SW1
03AB:38                    SEC            ;1 := 1-M
03AC:A5 CA                 LDA  1
03AE:E5 CE                 SBC  M
03B0:85 CA                 STA  I
03B2:A5 CB                 LDA  I+1
03B4:E5 CF                 SBC  M+1
03B6:85 CB                 STA  I+1
03B8:A5 CB                 LDA  I+1       ;BRANCH ON I<1
03BA:30 06                 BMI  INCJ
03BC:D0 B2                 BNE  RELAY3
03BE:A5 CA                 LDA  I
03C0:D0 AE                 BNE  RELAY3
03C2:E6 EB         INCJ    INC  J         ;J := J+1
03C4:D0 02                 BNE  INCJ2
03C6:E6 EC                 INC  J+1
03C8:A5 EC         INCJ2   LDA  J+1       ;BRANCH ON J>K
03CA:C5 D7                 CMP  K+1
03CC:90 A4                 BCC  RELAY2
03CE:A5 EB                 LDA  J
03D0:C5 D6                 CMP  K
03D2:90 9E                 BCC  RELAY2
03D4:18                    CLC
03D5:F0 9B                 BEQ  RELAY2
03D7:D0 9B                 BNE  RELAY1
```

MICRO

MICRObits *(continued)*

PET/CBM Owners

Real world software at low cost. 2114 RAM adapter and 4K Memory Expansion for "old" 8K PETs. Write for free catalog!

Optimized Data Systems
Dept. M, Box 595
Placentia, CA 92670

PET Arcade Software

Astroidz and *Munchman* games for your 8K old-new ROMS. *Astroidz* are invading the galaxy. Four levels of play. *Munchman* is based on arcade game Pac-Man. ZIP and ZAP are out to get you. Fantastic graphics. $9.95 each cassette.

ComputerMat
Box 1664M
Lake Havasu, AZ 86403

PASCAL LEVEL 1

This Pascal system allows the development of BRUNable programs. The system supports IF-THEN-ELSE, REPEAT-UNTIL, FOR-TO/DOWNTO-DO, WHILE-DO, CASE-OF-ELSE, FUNCTION, PROCEDURE, PEEK, POKE as well as disk I/O via DOS (specify 3.2 or 3.3). Price $35.00. Send SASE for more information.

On-Going Ideas
RD #1, Box 810
Starksboro, VT 05487

Extended SYM-BASIC

Adds 30 commands, requires 16K, $85 US/$95 Can., object on cassette, manual, and source listing. SYM-FORTH 1.0: fig-FORTH for 16K SYM-1. Editor, assembler, cassette interface $135 US/$155 Can., object on cassette, manual and source listing.

Saturn Software Limited
8246 116A St.
Delta, B.C., V4C 5Y9
Canada

MICRO

6502 COOKBOOK

For the machine language programmer who knows it is time to get serious! Here are search and sort routines, utilities, I/O & interrupt procedures, conversion routines, stack operations. Contains flow charts, source listings, explanations. Machine codes provided in both octal and hexadecimal. Tens of thousands of our programmer's cookbooks are in use.

Order publication: 6502 SOFTWARE COOKBOOK
Price in United States: just $12.95 + $1.00 s/h by mail.

Please include remittance with order. Allow 3 - 4 weeks for delivery. MasterCard & VISA credit cards accepted. Our phone line for credit card orders is (203) 888-1946. Foreign price list available. Write for more information.

☐ Check here for descriptive literature & catalog.

Name: ______________________________
Addr: ______________________________
City: ______________ State: ____ Zip: _____
MC/VISA # ______________ Bank: _____
Signature: ______________________________

SCELBI Publications
35 Old State Road, Oxford, CT 06483

POWER TO YOUR AIM

Treat your AIM to a quality power supply:

1. Designed to Rockwell's specifications for the AIM-65 (5 volts at 2 amps, regulated; 24 volts, .5 amps avg., 2.5 amps peak, unregulated).
2. *Overvoltage protection* to protect the expensive circuits in your AIM (5 volt output).
3. Handsome all metal case (two tone blue).
4. Fuse (externally accessible), switch, pilot light, line cord, cable from power supply to AIM — all included.
5. Conservative thermal design for long life.

$64.95 plus shipping (5 lbs.)

CA residents add 6 % sales tax.
VISA/MC, cashier's or registered check.
Personal check (allow 2 weeks to clear).

CompuTech
Box 20054
Riverside, CA 92516

Programming 6502 Assembly Language is no longer frightening or a monster problem. Because Randy Hyde has written *the* book that's *easy* to understand, *easy* to follow. It turns assembly language into the 'friendly language'. For anyone. For the average Apple II owner and the newest beginner.

Let Mr. Hyde get you started immediately, with string and math operations. See how to convert BASIC programs so they run up to 100 times faster! Discover Sweet-16, the 'hidden' 16-bit pseudo computer inside your Apple. Enjoy using your Apple to the maximum by following the step-by-step, practical examples. . .which turn you into a programmer in the blink of a chapter.

thanks to Mr. Hyde

$19.95 per easy-reading copy at computer stores everywhere, or from:

DATAMOST
19273 Kenya St.
Northridge, CA 91326
(213) 366-7160

VISA/MASTERCHARGE accepted.
$1.00 shipping/handling charge.
(California residents add 6% tax)

*Apple II is a trademark of Apple Computer, Inc.

Applesoft Memory Map Display

MEMAP is a short, exec file utility which creates memory maps of Applesoft programs without altering the memory contents.

N.D. Greene
Institute of Materials Science
University of Connecticut
Storrs, Connecticut 06268

Considerable memory space is needed for many engineering and business programs. Typically, these programs are long and use large data arrays with high-resolution graphics displays. Under these conditions, memory conflicts may occur and destroy the graphics picture, the program, or machine language routines. To prevent these problems during program development, it is necessary to know the location and length of the program and its variables. What's needed is a memory map. Although Apple Computer has published several memory maps (1-5), they are general and do not apply to specific programs. Also, some of the information is incorrect as noted by Peter Cook in an excellent two-part article in this magazine (6, 7).

There are several ways to map the Apple memory using the pointer addresses listed in table 1. These may be examined by immediate execution commands from the keyboard. For example, the end of a program can be determined by entering: PRINT PEEK (175) + PEEK (176)*256 followed by a carriage return. In a similar fashion all of the addresses listed in table 1 can be examined. However, this is rather tedious.

Another approach is to write a short program containing lines similar to the above example and append it to the end of the program being studied. This, of course, will lengthen the existing program and it is necessary to subtract the length of the appended portion to determine the original program length. A variation of this is to create a machine language program which examines and prints the contents of the pointer addresses. Machine language programs are "transparent;" they do not influence the pointer addresses. This is the method used by Cook (7) to develop memory maps. However, the machine language routine may overwrite the program in memory. If this occurs, the program will have to be re-entered after each memory examination.

The above difficulties can be avoided by using commands similar to the previous illustration *via* a disk exec file. If this file does not define variables or strings it can be used to examine memory locations without altering them.

The Program

Running the program shown in listing 1 will create an exec file, MEMAP. Once the file is created, it is activated by the command: EXEC MEMAP. MEMAP prints the contents of the pointer addresses listed in table 1 and calculates the amount of memory

Table 1

Applesoft Pointer Addresses*
(Decimal)

	Low Byte	High Byte
HIMEM	115	116
Strings (end)	111	112
Arrays (end)	109	110
Arrays (start	107	108
LOMEM (variables-start)	105	106
Program (end)	175	176
Program (start)	103	104

* See reference 2.

Listing 1

```
5 REM
          MEMAP EXEC CREATE
                 BY
             N.D. GREENE
                1981

10A$ = "MEMAP":D$ =  CHR$ (4):Q$
    =  CHR$ (34)
20 PRINT D$;"OPEN";A$: PRINT D$;
    "WRITE";A$
30 PRINT "HOME:VTAB 1:HTAB 10:?"
    Q$"APPLESOFT MEMORY MAP"Q$
40 PRINT "VTAB3:?"Q$"MAX MEMORY:
    "Q$",49152"
50 PRINT "VTAB5:?"Q$"           HI
    MEM:"Q$",PEEK(115)+PEEK(116)
    *256"
60 PRINT "VTAB6:?"Q$"STRINGS----
    ------------------>"Q$";PEEK
    (115)+PEEK(116)*256-(PEEK(11
    1)+PEEK(112)*256)"
70 PRINT "VTAB7:?"Q$" "Q$",PEEK(
    111)+PEEK(112)*256"
80 PRINT "VTAB8:?"Q$"FREE MEMORY
    ------------------>"Q$";PEEK
    (111)+PEEK(112)*256-(PEEK(10
    9)+PEEK(110)*256)"
90 PRINT "VTAB9:?"Q$" "Q$",PEEK(
    109)+PEEK(110)*256"
100 PRINT "VTAB10:?"Q$"ARRAYS---
    ------------------>"Q$";PE
    EK(109)+PEEK(110)*256-(PEEK(
    107)+PEEK(108)*256)"
110 PRINT "VTAB11:?"Q$" "Q$",PEE
    K(107)+PEEK(108)*256"
120 PRINT "VTAB12:?"Q$"VARIABLES
    ------------------->"Q$";PE
    EK(107)+PEEK(108)*256-(PEEK(
    105)+PEEK(106)*256)"
130 PRINT "VTAB13:?"Q$"
    LOMEM:"Q$",PEEK(105)+PEEK(10
    6)*256"
140 PRINT "VTAB14:?"Q$" "Q$",PEE
    K(175)+PEEK(176)*256"
150 PRINT "VTAB15:?"Q$"PROGRAM--
    ------------------->"Q$";PE
    EK(175)+PEEK(176)*256-(PEEK(
    103)+PEEK(104)*256)"
160 PRINT "VTAB16:?"Q$" "Q$",PEE
    K(103)+PEEK(104)*256:VTAB16"

170 PRINT D$;"CLOSE";A$
```

Figure 1: **MEMAP screen outputs obtainad (A) before and (B) after running the following progrem:**

```
10 DIM A (1000)
20 A$ = "TRIAL"
30 A$ = A$ + A$
```

```
(A)
          APPLESOFT MEMORY MAP

MAX MEMORY:      49152

        HIMEM: 38400
STRINGS----------------------->0
                 38400
FREE MEMORY------------------->36307
                 2093
ARRAYS------------------------>0
                 2093
VARIABLES--------------------->0
        LOMEM: 2093
                 2093
PROGRAM----------------------->44
                 2049
```

```
(B)
          APPLESOFT MEMORY MAP

MAX MEMORY:      49152

        HIMEM: 38400
STRINGS----------------------->10
                 38390
FREE MEMORY------------------->31278
                 7112
ARRAYS------------------------>5012
                 2100
VARIABLES--------------------->7
        LOMEM: 2093
                 2093
PROGRAM----------------------->44
                 2049
```

used by the program and its strings, arrays and variables. A few comments about the program listing may be helpful. The ASCII character code (34) for a quote mark is defined as a string in line 10 and later used to introducc quotes into the text file. Since exec files mimic keyboard input, tbey leave cursor marks on the screen as each command is executed. For aesthetic reasons, the program prints over these marks with a blank (line 70 for example). The question mark, ?, is a shorthand notation for the PRINT statement. It is used throughout this program.

Output

Figure 1 illustrates the screen displays obtained when MEMAP is exec'd before and after running a short program. (Note: it is not possible to directly print the output because of the backspacing and overprinting used. A screen dump routine is required to obtain a printed copy. This is of little consequence since permanent copies are rarely needed.) Programs start at location 2049 and progress upward. This program is located between 2049 and 2093 and is therefore 44 hytes long. Unless otberwise instructed, the computer sets LOMEM at the end of the program and fills the spaces above it with tbe program variables. These are not defined until the program is run as sbown by comparing figures 1A and 1B. This program has a single variable, A$. Each floating point variable requires seven bytes of memory

as confirmed by this map. Above the variable space, arrays are stored. Each floating point array requires seven bytes for indexing and five bytes for each array element. In this case, the total space needed is 7 + (1001 × 5) or 5012 bytes which is confirmed by figure 1B. Array space is not reserved until the program is run.

A 48K Apple has an upper memory capacity of 49152 bytes. The disk system (DOS) resets HIMEM to 38400 to protect its operating instructions. Below this point, redefined strings are stored; one byte for each string element. Line 30 in the example program redefines A$ as A$ + A$ which contains ten letters. Defined strings (e.g. line 20) are stored in the program area rather than in the so-called string region. The free or unused memory is located between the upper end of the array memory space and the bottom of the string storage area, as shown.

If several redefined strings are used in a program, each one is entered at HIMEM, pushing previously stored strings downward. If this is repeated, the new strings are added at HIMEM and the previous strings, pushed downward, are left as residuals or "garbage." This effect can be illustrated by modifying the program used in figure 3 to repeatedly create new strings (figure 4). Here two variables, A$ and I, are defined which require 2 × 7 or 14 bytes. The redefinition of A$ is repeated 2000 times, which consumes 20K of memory!

Applications

Memory maps are interesting by themselves, but their real use is to assist in program development. For example, consider the program and map shown in figure 2. The string "garbage" extends down to 18400, which is within the area normally used by high-resolution graphics, page two (16384 to 24575). Thus, it would not be possible to use HGR2 displays with this program without overwriting the graphics picture. This can be verified by adding a new line, 12 HGR2, and running the program. In a few moments, the Hi-Res screen fills with meaningless hash. However, figure 2 indicates that it should be possible to use page one of Hi-Res graphics, since it occupies the space between 8192 and 16383. Changing line 12 to HGR and running the program shows no evidence of overwriting: the screen remains black and clear.

Figure 2: MEMAP screen output obtained after running the following program:

```
10 DIM A (1000)
15 FOR I = 1 TO 2000
20 A$ = "TRIAL"
30 A$ = A$ + A$
40 NEXT
```

```
              APPLESOFT MEMORY MAP

MAX MEMORY:        49152

           HIMEM: 38400
STRINGS------------------------->20000
                   18400
FREE MEMORY--------------------->11260
                   7140
ARRAYS-------------------------->5012
                   2128
VARIABLES----------------------->14
           LOMEM: 2114
                   2114
PROGRAM------------------------->65
                   2049
```

Perhaps the most important aspect of using MEMAP in program development, is its educational value. In addition to showing that Hi-Res page one may be used without conflicts, figure 2 also suggests that redefined strings should be avoided or used sparingly. The string clearing effects of the FRE command can be seen by comparing before and after memory maps. Examining other programs with MEMAP suggests other ways to save space and/or to avoid memory conflicts. These include using integer rather than floating point arrays, employing multi-statement program lines, and moving HIMEM and LOMEM to protect parts of programs.

References

1. *Apple II Reference Manual*, Apple Computer Inc., 1978, p. 136.
2. *Applesoft II Basic Programming Reference Manual*, Apple Computer Inc., 1978, pp. 126, 127, 137.
3. *A Manual for Using the Apple Disk II with DOS Version 3.2*, Apple Computer Inc., 1979, pp. 140-142.
4. *The Applesoft Tutorial*, Apple Computer Inc., 1979, pp. 140-142.
5. *Apple II Reference Manual*, Apple Computer Inc., 1979, pp. 68-73.
6. Peter A Cook, *Apple Memory Maps, Part 1*, MICRO, No. 35, April 1981, pp. 27-35.
7. Peter A. Cook, *Apple Memory Maps, Part 2*, MICRO, No. 36, May 1981, pp. 45-56.

MICRO

Applesoft Line Finder Routine

This 55-byte machine language program will display the bytes constituting a specified line in an Applesoft program. This program also demonstrates how you can use the subroutines available in Applesoft and the Apple Monitor.

Peter J. G. Meyer
55 Sutter St., Suite 608
San Francisco, California 94104

The Applesoft Interpreter (at $D000-$F7FF) and the Apple Monitor ($F800-$FFFF) contain many useful machine language subroutines which may be utilized by programmers. Most of these subroutines are documented briefly in John Crossley's article "Applesoft Internal Entry Points" in the first issue of *The Apple Orchard*.

One such subroutine is named FNDLIN (at $D61A), and its task is to find the location, in memory, of a given line of an Applesoft program. To see why one might wish to do this, consider the following simple problem: how do you print "APPLE][PLUS" from within a program? This is easily reduced to two simpler problems: how to print "]" and "["? The former is available on the Apple keyboard in the guise of shift-M, but the latter is not enterable from the keyboard. A solution is to include in your Applesoft program the line PRINT "APPLE]Z PLUS", and then replace the hexadecimal number which represents 'Z' (namely, $5A) with the number which represents '[' (namely, $5B). This requires examination of the region of memory containing the tokenized form

Listing 1

```
                    ;******************
                    ;*                *
                    ;*  LINE FINDER   *
                    ;*                *
                    ;* BY PETER MEYER *
                    ;*                *
                    ;******************
                    ;
                    ;APPLESOFT SUBROUTINES
                    ;
                    FNDLIN EQU $D61A
                    ADDON  EQU $D998
                    REMN   EQU $D9A6
                    LINGET EQU $DA0C
                    CHKCOM EQU $DEBE
                    ;
                    ;MONITOR SUBROUTINES
                    ;
                    XAM    EQU $FDB3
                    BELL   EQU $FF3A
                    MONZ   EQU $FF69
                    ;
                    ;ZERO PAGE LOCATIONS
                    ;
                    A1     EPZ $3C
                    A2     EPZ $3E
                    LINNUM EPZ $50
                    LOWTR  EPZ $9B
                    TXTPTR EPZ $B8
                    ;
                           ORG $300          ;RELOCATABLE
                    ;
                           OBJ $800
                    ;
0300 20BEDE                JSR CHKCOM        ;CHECK FOR COMMA
0303 200CDA                JSR LINGET        ;GET LINE NUMBER
0306 201AD6                JSR FNDLIN        ;SEARCH FOR LINE IN BASIC PROGRAM
0309 B003                  BCS FOUND
030B 4C3AFF                JMP BELL          ;NOT FOUND
030E A59B           FOUND  LDA LOWTR         ;STORE STARTING ADDRESSS AT A1
0310 A49C                  LDY LOWTR+1
0312 853C                  STA A1
0314 843D                  STY A1+1
0316 A59B                  LDA LOWTR         ;SET TXTPTR TO STARTING
0318 18                    CLC               ; ADDRESS + 4
0319 6904                  ADC #$04
031B 85B8                  STA TXTPTR
031D A59C                  LDA LOWTR+1
031F 6900                  ADC #$00
0321 85B9                  STA TXTPTR+1
0323 20A6D9                JSR REMN          ;FIND END OF LINE
0326 2098D9                JSR ADDON         ;SET TXTPTR TO END OF LINE
0329 A5B8                  LDA TXTPTR
032B A4B9                  LDY TXTPTR+1
032D 853E                  STA A2            ;STORE ENDING ADDRESS AT A2
032F 843F                  STY A2+1
0331 20B3FD                JSR XAM           ;DISPLAY MEMORY FROM A1 TO A2
0334 4C69FF                JMP MONZ          ;ENTER MONITOR MODE
0337                ;
                          END
```

of the PRINT statement, locating the $5A, and replacing it with $5B. In the case of an Applesoft program composed of only a few lines, this can be done by direct inspection of memory using the Monitor. But, if your program has hundreds of lines, then another method is called for.

Given in listing 1 is a short, machine language program which is invoked (from BASIC command mode) by a statement of the form

```
CALL LOCATION, LINE
```

where LOCATION is the location (in decimal) of the machine language routine (it is relocatable), and LINE is the number of the line in the program to be searched for. If the routine finds the line, then it will display the bytes constituting the line and leave you in Monitor mode. (To return to BASIC command mode, enter Control-C.) If there is no line of the specified number in the Applesoft program, then the only result is a beep.

Suppose the routine is loaded or assembled at $300 (decimal 768), your Applesoft program is in RAM, and you wish to find the location of line 3370, which is, say, PRINT "]X". If you enter CALL 768,3370 then the bytes constituting the line will be displayed as follows:

```
xxxx- yy zz 2A 0D BA 22 5D 5A
      22 00
```

where xxxx is the address of the start of the line, yy zz is the pointer to the beginning of the next line (low-byte first), 2A 0D is the line number in hexadecimal (low-byte first), and 00 is the end-of-line token. The remaining five bytes are the tokenized form of the statement PRINT "]Z" (PRINT is represented by one byte: BA). If, for example, the address of the line is $1A92 then (from Monitor mode) you can enter:

```
1A99: 5B
```

which has the effect of replacing the byte '5A' with the byte '5B'. If (after Control-C-ing back to BASIC) the line is then LISTed, it will appear as PRINT "][", and will print accordingly.

For those readers without assemblers, the routine may be entered from Monitor mode by typing in 300: 20 BE DE 20 OC .... (See listing 1 for the remaining bytes.) Once entered, it may be saved to disk by entering BSAVE LINE FINDER, A$300, L$37. To use it, BLOAD LINE FINDER and proceed as above.

Apart from the utility, this routine is interesting because it relies almost entirely on subroutines in the Applesoft Interpreter and the Monitor, which is why it is only 55 bytes long. The five Applesoft subroutines and three Monitor subroutines which are used are given in listing 1 along with their addresses.

The routine works as follows: after you enter, e.g., CALL 768,3370, this statement is placed in the buffer (at $200) and the zero page pointer TXTPTR is set to the first byte (the token for CALL). Upon invocation of the routine at location 768, TXTPTR is pointing to the comma, and the subroutine CHKCOM checks for this. (If there is no comma, a syntax error message results.) The routine then gets the line number using the subroutine LINGET, and places this (in hexadecimal form, low byte first) at LINNUM. The subroutine FNDLIN picks up this number and searches the Applesoft program for the line so numbered. If it does not find such a line, it returns with the carry flag clear. In this case the routine sounds the bell and returns to BASIC command mode.

If FNDLIN finds the line, then it returns with the carry flag set. It then deposits the address of the line at LOWTR (low byte first, as usual). The routine stores this address at AI, for later use by the subroutine XAM (eXAMine memory), which will display the bytes constituting the line.

Having found the address of the beginning of the line, the subroutines REMN and ADDON are used to find the address of the end. In order to use the subroutine REMN, which searches from the byte pointed to, by TXTPTR, until it finds an end-of-line token (00), the routine first sets TXTPTR to four places past the beginning of the line. This is to skip the link pointer and the line number, since the line number may contain 00 (as in 0A 00, representing 10), which would mislead REMN. REMN is then invoked, and returns

with the offset to the end-of-line in the Y register. ADDON adds this offset to TXTPTR, so that TXTPTR is then pointing to the end of the line. This address is stored at A2, and XAM is invoked to display the bytes from A1 to A2.

Readers wishing a fuller understanding should consult the aforementioned article by John Crossley, and the Apple manual entitled *Apple II Monitors Peeled*, for details of the subroutines given in listing 1.

While studying mathematics and philosophy in the late 1960's, Peter Meyer wrote programs in FORTRAN for scientific and technical applications. He acquired an Apple in early 1980 and proceeded to develop the memo program *Agenda Files* (Special Delivery Software). Currently he is studying the internals of Applesoft, and is designing a system for interfacing Applesoft programs with machine language subroutines.

MICRO

PET Vet *(Continued from page 35)*

user "stack," which is a series of memory locations used for most language operations and much data storage.

A pointer to the next available location is constantly maintained as values and addresses are PUSHed onto it or POPped from it. Most operations act on one or both of the top two entries, removing them and replacing them with the result of the operation. For instance, to add 5 and 3, you would first PUSH a 5 onto the stack, then PUSH a 3, and finally invoke the + operation which would POP both the 5 and the 3 and leave only the result of the operation on the stack: an 8.

A program to fill the screen with PET characters. The BASIC version runs considerably slower than the RPL version.

Economy BASIC Version

```
10 FORI=0TO999:POKE32768+I,IAND255:NEXT
```

Source for RPL Version

```
10 33767 32768 FOR FN # POKE NEXT
```

RPL provides all of the necessary stack operations, including ones that allow rolling an entry to the top from a specified depth and interchanging the top two entries. It allows conditional branching, Boolean operations, PEEKs and POKEs and their 16-bit equivalents, subroutines, nested FOR... NEXT loops, nested IF...THEN...ELSE constructions, random numbers, GET, INPUT and even the dreaded GOTO. Character string manipulation and printing take a little extra effort, but are straightforward. The RPL operations actually end up allowing more flexibility.

Surely there must be some things missing. But of course! Numbers may only be 16-bit integers (−32768 to 32767, or 0 to 65535), although routines to handle floating point numbers and larger integers could certainly be written. The built-in file-handling capabilities of BASIC are not duplicated. Also, all of the higher mathematical functions, like trig functions, square roots, and such, are lacking.

Line numbers are used only in editing and error detection. They have no meaning in the program flow. Instead you use symbols to label parts of your program. Symbols may be defined globally or locally, making the development of a subroutine library very easy. It is also possible to use symbolic constants, which can be defined at "compile time."

One big advantage of RPL is that it uses the PET editor, so you don't have to get used to a different, less powerful editor. BASIC and the PET's machine language monitor are available while RPL is present, so it is easy to load and save both source and object files. Inclusion of machine language routines for the ultimate in speed and economy of space is also very easy, especially with the forthcoming Samurai assembler which will use the same symbol table structure as RPL. Interfacing to BASIC is possible too, but the process is a little more involved.

If you have never spent much time with H-P calculators or FORTH, stack manipulation might be a little confusing. Samurai Software has available a program called "SIM," a symbolic debugger for programs written in RPL, which can be included on the same disk or cassette with RPL. Not only does this allow stepping through a program, setting breakpoints, PUSHing, PULLing, and setting the PC, but it also shows each operation before it is executed and then shows the results of the contents of the stack. This program is almost essential for debugging, since the only runtime error message is "P!" to indicate stack over- or underflow. It also illustrates the intricacies of stack manipulation very nicely.

The documentation is about the best I have ever seen. The manual begins with enough information to get anyone programming quickly, followed by a section on more advanced techniques. (There are appendices with other information.)

RPL is not a standard language, although it has more in common with FORTH than any other. Instead of the "threaded" structure of FORTH, RPL uses a P-code structure like Pascal. RPL is generally faster and more conservative of memory than FORTH. FORTH can be applied on nearly every computer, while you can use RPL only on a CBM/PET. FORTH's portability has a cost, in that routines that already exist in the PET's ROMs (or any other machine's operating system) must be duplicated, thus eating up valuable memory.

RPL will serve well the need for a language that is faster than BASIC yet easier to program than assembly language. The package is well-thought-out and well-documented. RPL is more difficult to program than BASIC and more difficult to read, but it does have many elements of the structured languages like Pascal. Its intimacy with the PET operating system is an advantage over FORTH in speed and memory conservation, but it makes it impossible to run on a non-PET.

RPL is available from Samurai Software (P.O. Box 2902, Pompano Beach, FL 33062) for $49.95 on disk or $44.95 on cassette. Specify your ROM and disk drive types.

Commodore Announces New Products

Six new books dealing with various aspects of the new SuperPET system are now available:

1.	System Overview	$ 5.95
2.	Waterloo microAPL	$ 9.95
3.	Waterloo microBASIC	$10.95
4.	Waterloo microFORTRAN	$10.95
5.	Waterloo microPascal	$10.95
6.	Waterloo 6809 Assembler	$10.95

A new high-speed printer, the CBM 8023P, is now available for $995. It features bidirectional printing at 150 cps in up to 136 columns, and both tractor and friction feeds. Upper and lower case are available, as well as PET graphic characters and user-defined characters. There is also RAM available for formatting data.

MICRO

Applesoft and Matrices

This machine language program performs the most commonly used special matrix operations, as well as most Applesoft operations. The program can be linked to Applesoft by means of the & statement. Two advantages of using this program rather than a BASIC subroutine are a significant increase in execution speed (on the average a factor 5) and greater convenience. The required system configuration for the program is a 48K Apple with Applesoft in ROM (or in the Language Card).

Cornelis Bongers
Erasmus University
Postbox 1738
3000 DR Rotterdam,
The Netherlands

For those who are not accustomed to working with matrices, a matrix is a block of numbers. Several operations can be performed on a matrix or a pair of matrices. For instance, adding two matrices A and B together, we obtain a matrix C, whose elements consist of the sums of the corresponding elements of A and B. Thus if,

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 1 & 4 \\ 4 & -2 & 1 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 2 & 4 & 7 \\ 1 & 8 & -6 \\ 5 & 0 & 1 \end{bmatrix}$$

then the sum of A and B is

$$C = \begin{bmatrix} 3 & 7 & 12 \\ 3 & 9 & -2 \\ 9 & -2 & 2 \end{bmatrix}$$

It will be clear that A, B, and C can be represented by three 2-dimensional arrays in BASIC. When A and B have to be added, the following BASIC routine may be used:

```
100 FOR I=1 TO N: FOR J=1
    TO M: C(I,J)=A(I,J)+B(I,J):
    NEXTJ,I
```

where N and M are both equal to 3 in our example. When using the machine language program, this routine can be replaced by the statement:

```
100 & C=A+B
```

Note that by using the latter statement, the names of the matrices are irrelevant. In the BASIC routine the names of the matrices always must be A, B, and C to comply with the names of the BASIC arrays.

Applesoft Operations

Except for comparison, SCRN(, and CHR$, all the Applesoft operators and functions that can be used on real variables or expressions are available for matrix operations. There are, however, some restrictions on the syntax of the matrix statement. First, no more than 3 matrices may be used in a matrix statement. Second, single-valued expressions (or variables) must be put between brackets. Another restriction is that matrices used in an & statement must have two dimensions. Each of these dimensions must be larger than 0 and smaller than 255. Furthermore, each matrix appearing in an & statement must have been dimensioned previously by means of a DIM statement. For the exact syntax of the matrix statement we refer to the 'Instructions' section of the article. Some examples are listed below.

Example 1:

```
10 DIM A(10,10): B=1
20 &A=(B): A=RND(A):
   A=A*(10): A=INT(A)
```

In this example, the array A is set equal to 1. Next, the RND function is performed on all elements of A, so that A now contains random numbers between 0 and 1. Then A is multiplied by 10, and the INT function is executed on each element of A. After the execution of line 20, A is thus filled with random numbers between 0 and 9. Note that the statement A=(RND(1)) puts all elements of A equal to the same random number.

Example 2:

```
10 DIM A(5,6), B(5,6), C(5,6)
20 B=3
30 &A=(3): B=(2): C=A*B:
   C=C^(B)
```

The statement C=A*B multiplies the corresponding elements of A and B and stores the result in the corresponding elements of C. After the execution of this statement, all elements of C are therefore equal to 6. Note that for a successful execution of the statement, A, B, and C must have the same dimension (or order). By means of the last statement, all elements of C are raised to the third power. If, instead of the statement C=C^(B), the statement C=C^B is used, all elements of C will become equal to the second power of 6, because now the *matrix* B instead of the *variable* B is taken.

Matrix Operations

Although the operations and functions used in the examples above can be handy sometimes, they hardly justify the writing of a machine language program. The real usefulness of the program is, therefore, not its ability to perform Applesoft functions and operations, but rather to handle some specific

matrix operations as well. The following operations are implemented:

1. A=IDN(*aexpr*) where A must be a square matrix and 1<=*aexpr*<=N if N is the order of A. This statement puts A equal to a matrix consisting of zeros and ones. If *aexpr* equals one, A becomes the identity matrix. For larger values of *aexpr*, the columns of the identity matrix will be rotated *aexpr* − 1 positions to the left. For instance, if A and B are square matrices of order 3, then A=IDN(1) and B=IDN(2) return.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

2. A=TRN(B) puts A equal to the transpose of B. If B is of order p by q, then A must be of order q by p. Putting a matrix equal to its own transpose (i.e. A=TRN(A)) is not allowed. For instance, if B equals,

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 0 \end{bmatrix}$$

then A=TRN(B) will return

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 0 \end{bmatrix}$$

3. A=B.C puts A equal to the matrix product of B and C. If B is of order p by q, then the first dimension of C must equal q. In case the second dimension of C equals r (thus C is q by r), the matrix A has to be of the order p by r. Furthermore, the matrix on the left of the "=" sign may not equal one of the matrices on the right of the "=." As an example, we can multiply the matrices A and B in the example above by means of the statement &C=A.B. This leads to

$$C = \begin{bmatrix} 35 & 14 \\ 14 & 20 \end{bmatrix}$$

4. A=MIN(B), A=MAX(B) or A=ABM(B) put A respectively equal to the minima, the maxima, or the absolute maxima of the columns of B. The overall maximum, minimum, or absolute maximum of B is stored in A(0,1). If B is of order p by q, then A must be of order q by 1.

5. A=INV(B) puts A equal to the inverse of B and stores the determinant of B in A(0,0). A and B must be square and of the same order. The statement D=INV(C), where C equals the matrix above, returns for instance,

$$D = \begin{bmatrix} .0396825397 & -.0277777778 \\ -.0277777778 & .0694444444 \end{bmatrix}$$

At the execution of the inverse statement, values stored in the 0th row of the target matrix will be destroyed since this row is used to store some pointers. To obtain the inverse of a matrix A, the statement A = INV(A) also may be used. Finally, zeros on the main diagonal of the matrix to be inverted are allowed.

6. A=NEINV(B) gives the same result as A=INV(B) except that the program continues if a division by zero occurs when B is singular. When using NEINV, it is recommended to check the determinant of B (in A(0,0)) after execution of the statement. When B is singular, the determinant will be zero.

7. A=PNT (*aexpr*) displays the matrix A. For each element of A, *aexpr* positions are reserved, and a carriage return is generated after each row. If *aexpr* equals zero, the elements of A are separated by a blank.

An Application

An interesting application of matrix algebra is the linear model. The linear model can be used to analyze the influence of a number of variables, called the independent variables, on another variable, called the dependent variable. The model has the form,

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots b_m x_m + u,$$

where y denotes the dependent variable, and x_1, x_2, etc., denote the independent variables.

The last term, u, represents the influence of factors that were not included in the model. Usually this term is called the residual. As an example, suppose that we want to establish the relationship between the annual regional sales of a particular product (y), the number of times advertised (x_1) and the number of people living in the region (x_2). The available data are given in the table below.

Obs. No.	Y Sales	X_1 Advert.	X_2 Popul.
1	118	8	583
2	138	9	692
3	104	5	1082
4	65	1	836
5	46	1	628
6	61	2	244
7	48	1	632
8	66	2	172
9	78	5	319
10	69	2	383

In matrix algebra the model can be written as,

$$Y = X.B + U,$$

where B (the unknown coefficients) is of order 3 by 1 and Y (the sales), and U (the residuals) are of order 10 by 1. The matrix X is of order 10 by 3. The elements of the first column of X are equal to one (to account for b_0) whereas the second and third columns correspond to the columns under the heading $\mathbf{X_r}$, and $\mathbf{X_2}$ in the table. To fit the equation to the data, the least squares principle is used, which means that the coefficients are chosen such that the sum of the squares of the elements of U is minimized. This leads to the following solution for B,

$$B = (X'.X^{-1})X'.Y$$

where X' denotes the transpose of X. A BASIC program to compute the least squares solution is presented in listing 1, with the results of the example. The least squares equation shows that the sales increase by 9.5 for each additional advertisement (other things being equal) whereas an increase of 100 in the population of the region increases the sales by 1.6 (other things being equal).

The application given in this section was kept simple purposely. The linear model, for instance, can easily be extended with a tremendous amount of statistics which may (or may not) simplify the analysis of the data. Also the application presented gives only a narrow view on the wide field of problems in which matrix algebra may be useful. Examples include computations with Markov-type problems and the location of the maximum (or minimum) of a function of several variables by means of the Newton method.

The Machine Language Program

The hex dump of the program is presented in listing 2. As can be seen, it

is about \$700 bytes long and starts at \$8900. The end is at \$8FF2, which means that the area \$9000-\$9600 is free for other routines.

After the hex dump has been keyed in and saved, the program can be connected to an Applesoft program by means of the command : BRUN program name or, if you don't have a disk, by the monitor command : 8900 G. In the latter case you must enter Applesoft *via* the warm start (i.e., Control-C). The BRUN or 8900 G command executes the initialization routine at the start of the program that sets HIMEM to the appropriate value and installs the & vector. In case the & vector is destroyed during execution of a program, the matrix program can be reconnected by the command CALL 35072.

The program extensively uses zero page locations to increase execution speed. However, as a consequence, the ON ERR flag will be temporarily cleared during the execution of an & line since the matrix routines use the storage space of the ON ERR pointers. After the execution of the & line, the ON ERR flag and pointers are restored to their original values. Apart from zero page locations, the control Y and the & vector are used, which implies that values stored at \$3F5 - \$3FA will be destroyed.

In Case of an Error

If the interpreter returns an error message during the execution of an & line, there is either a bug in your statement or a bug in my program. In the first case, the error is probably caused by the violation of one of the following conditions:

1. Only matrices containing reals are allowed in the & line.

2. Matrices used in an & statement must have 2 dimensions.

3. Each dimension of a matrix must be larger than 0 and smaller than 255.

4. The orders of the matrices should satisfy the conditions in the "instructions" section of this article.

5. Each matrix appearing in an & statement must have been dimensioned earlier in the program by a DIM statement.

6. ON ERR doesn't work during the execution of an & line.

Although the other case (i.e. a bug in my program) seems at this time highly improbable to me since the program was heavily tested for several months, I am well aware that there are some kinds of bugs that can, as it seems, only be discovered by other people. Therefore, if you find one, I would appreciate it very much if you let me know.

Finally, a utility package which contains, among others, the matrix program, will be released soon. This utility package resides in the second 4K bank of the Language Card, and it will use only \$300 bytes of 'normal' RAM.

Instructions

This section contains the matrix expressions that can be executed by means of the & line. The syntax of the line is:

    & matrix expression: matrix
    expression: etc.

The following operators and functions may be used:

    operator := +, −, *, /, ^
                AND,OR
    function := SGN, INT, ABS,
                USR, FRE, PDL, POS,
                SQR, RND, LOG, EXP,
                COS, SIN, TAN, ATN,
                PEEK

Unless stated otherwise, matrices appearing in an & statement must have the same order, and matrix names on the left of the "=" sign can be chosen equal to matrix names on the right of the "=". The matrix expressions that are allowed are listed below.

I. Applesoft Operations and Functions with:

1.1 1 matrix and 1 expression
A = *(aexpr)*

Example:
A = (−1/2), B=(Z%)

1.2 2 matrices
A = B
A = −B
A = NOT B
A = function(B)

Example:
A = SIN(B)

1.3 2 matrices and 1 expression
A = B operator *(aexpr)*

Example:
A = B ^(COS(3))

1.4 3 matrices
A = B operator C

Example:
A = B/C

II. Specific Matrix Operations

2.1 A = IDN*(aexpr)* — Identity: A must be square and 1<= *aexpr* < = order of A.

2.2 A = TRN(B) — Transpose: if B is of order p by q, then A must be of order q by p. A=TRN(A) is not allowed.

2.3 A = B.C — Multiplication: if B is of order p by q and C of order q by r, then A must be of order p by r. A=A.C or A=C.A is not allowed.

2.4 A = MIN(B), A = MAX(B), A = ABM(B) — Minimum, maximum or absolute maximum: if B is of order p by q then A must be of order q by 1. After execution A(0,1) contains the overall minimum, maximum or absolute maximum of B.

2.5 A = INV(B) — Inverse: A and B must be square and of the same order. After execution, A(0,0) contains the determinant of B.

2.6 A = NEINV(B) — Inverse: same as INV, except that singularity of B doesn't stop the program.

2.7 A = PNT*(aexpr)* — Print: if *aexpr*=0 the elements are separated by a blank, else *aexpr* positions are reserved for each element.

Cornelis Bongers is an assistant professor of statistics at the Erasmus University in Rotterdam, The Netherlands. He uses his Apple for solving statistical problems, such as likelihood maximization and the estimation of the parameters of density functions. As a hobby, he develops machine language utility programs for the Apple to extend Applesoft, via the & instruction, with several functions that are not implemented, such as PRINT USING, Sort and Storing and Recalling arrays to or from disk.

Listing 1

```
5  REM  THE LINEAR MODEL
10  HOME
20  INPUT "NUMBER OF OBSERVATIONS ? ";N
30  INPUT "NUMBER OF INDEPENDENT VARIABLES ? ";M:M1 = M + 1
40  IF M1 >  = N THEN  PRINT : PRINT "TOO FEW OBSERVATIONS ": STOP
50  DIM X(N,M1),XA(M1,N),Y(N,1),B(M1,I),E(N,1),EA(I,N),S(MI,MI)
60  DIM VI(1,1),V2(1,I),H(M1,1),J(1,N)
70  PRINT : PRINT "INPUT THE ELEMENTS OF THE Y-VECTOR": PRINT
80  FOR I = 1 TO N
90  PRINT "ELEMENT ";I;" ? ";: INPUT "";Y(I,I):X(1,I) = 1
100  NEXT 1
110  FOR J = 2 TO M1
120  PRINT : PRINT "INPUT THE ELEMENTS OF THE X";J - I;" VECTOR": PRINT
130  FOR 1 = 1 TO N
140  PRINT "ELEMENT ";I;" ? ";: INPUT "";X(I,J)
150  NEXT 1,J
160  REM  CALCULATE RESULTS
170  & XA = TRN(X):S = XA.X:S = NEINV(S):H = XA.Y:B = S.H
180  IF S(0,0) = 0 THEN  PRINT "THE S-MATRIX IS SINGULAR": STOP
190  PRINT : PRINT "THE LEAST SQUARES EQUATION EQUALS ": PRINT
200  PRINT "Y = ";B(1,1);
210  FOR J = 2 TO M1: IF B(J,I) >  = 0 THEN  PRINT "+";
220  PRINT B(J,1);"*X";J - 1;
230  NEXT : PRINT : PRINT
240  & E = X.B:EA = TRN(E):E = Y - E
250  PRINT "** THE TABLE OF RESIDUALS **": PRINT
260  PRINT "NO"; TAB( 4);"OBSERVED Y"; TAB( 16);"ESTIMATED Y"; TAB( 29);"RESIDUAL"
270  FOR I = 1 TO N
280  PRINT I; TAB( 4);Y(I,I); TAB( 16);EA(1,I); TAB( 29);E(1,I)
290  NEXT I: PRINT
300  & EA = TRN(E):V1 = EA.E
310  PRINT "STANDARD DEV. RESIDUALS: "; SQR (V1(1,1) / (N - M1))
320  & J = (1):V2 = J.Y:V2 = V2 / (N):E = Y - (V2(I,1)):EA = TRN(E):V2 = EA.E
330 R = (V2(1,1) - VI(I,1)) / V2(1,1): IF R < 0 THEN R = 0
340  PRINT "R^2";: HTAB (24): PRINT ": "; SQR (R)
350  END
```

Output of the Exampla

```
THE LEAST SQUARES EQUATION EQUALS

Y = 35.9942408+7.54687973*X1+.0160419893*X2

** THE TABLE OF RESIDUALS **

NO OBSERVED Y  ESTIMATED Y  RESIDUAL
1  118         121.721758   -3.72175834
2  138         133.017215   4.98278511
3  104         101.086072   2.91392821
4  65          58.9522235   6.04777649
5  46          55.6154898   -9.61548973
6  61          59.0022456   1.99775441
7  48          55.6796577   -7.6796577
8  66          57.8472224   8.15277764
9  78          88.846034    -10.846034
10 69          61.2320821   7.7679179

STANDARD DEV. RESIDUALS: 8.31192607
R^2                    : .971018065
```

(Continued on next page)

***Listing 2:* Hex Dump**

```
8900- A9 00 85 73 A9 89 85 74
8908- A9 73 8D F6 03 A9 89 8D
8910- F7 03 60 A8 49 44 CE 50
8918- 4E D4 4E 45 49 4E D6 49
8920- 4E D6 54 52 CE 4D 49 CE
8928- 4D 41 D8 41 42 CD 00 8D
8930- 8E 8C F1 8D EC 8D EF 8C
8938- FC 8B 9C 8B 9F 8B A2 00
8940- 00 00 00 00 A9 8A 48 A9
8948- CB 48 A9 11 8D F9 03 A9
8950- 8A 8D FA 03 A2 F8 86 08
8958- 20 67 8A A9 D0 20 C0 DE
8960- C9 80 90 5E C9 C6 D0 36
8968- A0 18 20 2D 8A 20 67 8A
8970- 4C 0D 8A A2 06 B5 F3 48
8978- CA D0 FA A5 D8 48 86 D8
8980- 20 44 89 20 B7 00 D0 5C
8988- C9 00 D0 0C 68 85 D8 A2
8990- FA 68 95 FA E8 D0 FA 60
8998- 20 B1 00 4C 80 89 A0 15
89A0- C9 C9 F0 C6 C9 D2 90 3C
89A8- C9 E3 B0 38 29 7F 0A A8
89B0- 20 B1 00 B9 DC CF 8D F9
89B8- 03 B9 DD CF 8D FA 03 4C
89C0- 04 8A A2 FF 86 D7 E6 D7
89C8- A0 FF 38 C8 E8 BD 13 89
89D0- F1 B8 F0 F7 E9 80 F0 0F
89D8- BD 13 89 F0 35 C9 80 B0
89E0- E5 E8 90 F4 4C C9 DE 98
89E8- A2 B8 20 4A 8B A5 D7 F0
89F0- 68 0A AA 68 68 BD 2D 89
89F8- 48 BD 2E 89 48 20 B1 00
8A00- E0 05 90 55 20 BB DE 20
8A08- 67 8A 20 B8 DE A2 80 86
8A10- D7 60 20 67 8A F0 F6 C9
8A18- 2E D0 26 68 68 A9 8C 48
8A20- A9 6F 48 20 B1 00 20 67
8A28- 8A A2 01 D0 E2 18 B9 B3
8A30- D0 69 01 8D F9 03 B9 B4
8A38- D0 69 00 8D FA 03 4C B1
8A40- 00 38 E9 C8 C9 07 B0 9C
8A48- 85 D7 0A 65 D7 A8 20 2D
8A50- 8A C9 28 D0 D1 A9 40 85
8A58- D7 20 67 DD A2 3F 86 FE
8A60- A0 89 84 FF 4C 2B EB A9
8A68- 40 85 14 20 E3 DF A6 08
8A70- A5 12 05 11 F0 03 4C 76
8A78- DD 85 14 A0 04 A9 02 D1
8A80- 9B D0 46 C8 B1 9B D0 41
8A88- C8 B1 9B C9 02 90 3A 95
8A90- FC E8 C8 C0 09 D0 ED 98
8A98- 86 08 A2 9B 20 4A 8B A6
8AA0- 08 95 00 A5 9C 95 01 B5
8AA8- FB A2 CE 20 54 8B A6 08
8AB0- A0 CE 20 38 8B E0 FA D0
8AB8- 0D A2 01 B5 CE 95 06 B5
8AC0- 9B 95 1E CA 10 F5 4C B7
8AC8- 00 4C 96 E1 A5 D7 F0 10
8AD0- A2 01 4A 90 02 A2 03 B5
8AD8- F4 D5 F6 D0 EC CA 10 F7
8AE0- A2 05 B5 FA 95 17 CA 10
8AE8- F9 A5 F4 85 09 C6 09 F0
8AF0- 56 A5 F5 85 08 A2 17 20
8AF8- 48 8B A5 D7 F0 25 24 D7
8B00- 70 07 30 0C A2 1B 20 48
8B08- 8B A5 1B A4 1C 20 F9 EA
8B10- A2 19 20 48 8B A5 19 A4
8B18- 1A 20 E3 E9 24 D7 10 03
8B20- 20 53 EB C6 08 F0 C6 A5
8B28- 9D 20 F8 03 A6 17 A4 18
8B30- 20 2B EB 4C F5 8A A0 06
8B38- 18 B5 00 79 00 00 95 00
8B40- B5 01 79 01 00 95 01 60
8B48- A9 05 18 75 00 95 00 90
8B50- 02 F6 01 60 95 00 A0 00
8B58- 94 01 0A 36 01 0A 36 01
8B60- 90 E9 90 08 A0 05 84 FC
8B68- A0 00 F0 06 A4 06 84 FC
8B70- A4 07 84 FD 60 8A 48 84
8B78- AD A0 00 84 AE A4 06 84
8B80- 64 A5 07 20 B6 E2 18 8A
8B88- 65 9B 85 1E 98 65 9C 85
8B90- 1F 68 A2 A8 20 54 8B A2
8B98- 1E A0 A8 D0 9B A0 80 2C
8BA0- A0 00 2C A0 40 A2 00 86
8BA8- A5 A5 F6 C5 F5 D0 5C A6
8BB0- F4 E0 02 D0 56 CA B5 F6
8BB8- 95 F4 B5 FA 95 17 B5 CE
8BC0- 95 06 CA 10 F1 84 D7 A4
8BC8- A5 84 A7 C8 A2 00 20 75
8BD0- 8B A2 A8 A5 A5 20 54 8B
8BD8- 4C E7 8B A5 D7 4A B0 17
8BE0- A6 17 A4 18 20 2B EB A2
8BE8- 17 20 48 8B A9 FF 20 40
8BF0- EC 85 9D 45 D7 85 A2 E6
8BF8- A7 A5 A7 C5 F4 90 23 A5
8C00- D7 4A 90 0A 60 A5 F4 C5
8C08- F5 F0 F9 4C 96 E1 4A B0
8C10- F3 A2 01 E6 D7 B5 FA 95
8C18- 1E 95 17 CA 10 F5 A5 F4
8C20- 85 F5 A2 1E A0 A8 20 38
8C28- 8B A5 A5 85 A6 90 09 24
8C30- D7 50 05 68 A0 01 91 1E
8C38- A2 1E 20 48 8B E6 A6 A5
8C40- A6 C5 F5 B0 96 24 D7 50
8C48- 09 A0 01 B1 1E 48 29 7F
8C50- 91 1E A5 1E A4 1F 20 B2
8C58- EB F0 D4 45 D7 10 D0 A5
8C60- 1E A4 1F 20 F9 EA A5 A6
8C68- 85 08 A5 A7 85 09 D0 BF
8C70- A5 F4 C5 F8 D0 95 A5 F5
8C78- C5 F7 D0 8F A2 02 20 41
8C80- 8D A5 F6 C5 F9 D0 84 C6
8C88- F4 F0 66 A5 F5 85 CE A2
8C90- 03 B5 FC 95 17 CA 10 F9
8C98- 30 32 A9 8C 48 A9 B3 48
8CA0- 20 10 DE A5 F8 A4 F9 20
8CA8- F9 EA A5 FE A4 FF 20 7F
8CB0- E9 4C 47 DE 20 C1 E7 A2
8CB8- F8 20 36 8B A2 FE 20 48
8CC0- 8B C6 CF D0 D5 A6 FA A4
8CC8- FB 20 2B EB A2 FA 20 48
8CD0- 8B C6 CE F0 B2 A2 17 20
8CD8- 48 8B 85 F8 A5 18 85 F9
8CE0- A5 19 85 FE A5 1A 85 FF
8CE8- A5 F6 85 CF 20 4E E8 F0
8CF0- CB 60 20 08 E1 A5 A1 85
8CF8- D7 A9 FF D0 10 A5 F4 C5
8D00- F7 D0 82 20 44 8D A5 F5
8D08- 38 E5 F6 D0 F4 85 08 A5
8D10- 08 10 03 20 8E FD C6 F5
8D18- F0 D7 A5 F4 85 F8 A2 FA
8D20- 20 48 8B 85 FE A5 FB 85
8D28- FF A2 FC 20 48 8B C6 F8
8D30- F0 DD A5 08 30 1F A0 04
8D38- B1 FC 91 FE 88 10 F9 30
8D40- 3D 20 46 8D A2 00 A5 FA
8D48- D5 FC D0 A5 A5 FB D5 FD
8D50- D0 9F 4C 30 E4 A5 FE A4
8D58- FF 20 F9 EA 20 34 ED 20
8D60- E7 E3 20 00 E6 85 F9 38
8D68- A5 D7 AA F0 08 E5 F9 90
8D70- 14 F0 06 AA CA E8 20 4A
8D78- F9 A5 F9 20 40 DB A2 FE
8D80- 20 36 8B 90 A4 A9 AA 20
8D88- ED FD CA D0 FA F0 EF 20
8D90- 05 8C 20 08 E1 A5 A0 F0
8D98- 03 4C 99 E1 85 9D 85 D7
8DA0- A5 A1 F0 F5 C5 F4 B0 F1
8DA8- 85 FF 20 E0 8A A5 1E 85
8DB0- 9B A5 1F 85 9C A6 FF CA
8DB8- A0 00 20 75 8B A2 06 20
8DC0- 48 8B A9 06 A0 F1 20 F9
8DC8- EA A2 1E 20 36 8B C6 F4
8DD0- F0 1A A6 1E A4 1F 20 2B
8DD8- EB E6 FF A5 F5 C5 FF D0
8DE0- E8 A2 1E 20 48 8B 20 48
8DE8- 8B 4C CE 8D 60 A0 00 2C
8DF0- A0 01 84 FE 20 05 8C A5
8DF8- FA C5 FC D0 06 A5 FB C5
8E00- FD F0 07 A0 80 84 D7 20
8E08- CC 8A A2 01 B5 1E 95 9B
8E10- 95 F8 95 FA 95 F6 CA 10
8E18- F3 A9 06 A0 F1 20 F9 EA
8E20- A6 9B A4 9C 20 2B EB A5
8E28- F5 85 1D 38 A5 F5 E5 1D
8E30- 85 A5 C6 1D D0 32 A0 01
8E38- 84 FF A4 F5 88 84 1D C6
8E40- 1D F0 A9 38 A5 FA E5 06
8E48- 85 FA A5 FB E5 07 85 FB
8E50- A5 F8 E9 05 85 F8 B0 02
8E58- C6 F9 20 89 8F A0 01 A9
8E60- 00 91 FA 88 91 FA F0 D7
8E68- A2 F8 20 48 8B A2 FA 20
8E70- 36 8B A2 F6 20 36 8B 20
8E78- 48 8B A4 1D 88 F0 1E A0
8E80- 41 20 C5 8B A0 01 38 B9
8E88- 08 00 AA 65 1D E5 F5 F0
8E90- 01 8A 91 FA 88 10 EF C8
8E98- 84 FF 20 89 8F A5 9B A4
8EA0- 9C 20 F9 EA A5 F6 A4 F7
8EA8- 20 7F E9 A6 9B A4 9C 20
8EB0- 2B EB 05 FE F0 8B A5 F6
8EB8- A4 F7 20 F9 EA A9 06 A0
8EC0- F1 20 66 EA A6 F6 A4 F7
8EC8- 20 2B EB A2 FF 38 A5 FA
8ED0- A4 FB 20 48 8F A5 F5 85
8ED8- 09 A6 9B A4 9C 86 1B 84
8EE0- 1C A6 F8 A4 F9 86 17 84
8EE8- 18 A2 1B 20 36 8B A2 17
8EF0- 20 36 8B C6 09 F0 45 A4
8EF8- 09 C4 1D F0 EC A5 F5 85
8F00- 08 A6 FA A4 FB 86 19 84
8F08- 1A A2 1B 20 48 8B A2 19
8F10- 20 48 8B C6 08 F0 D7 A4
8F18- 08 C4 1D F0 EC A5 19 A4
8F20- 1A 20 F9 EA A5 17 A4 18
8F28- 20 7F E9 A5 1B A4 1C 20
8F30- BE E7 A6 1B A4 1C 20 2B
8F38- EB 4C 09 8F A2 00 A5 F8
8F40- A4 F9 20 48 8F 4C 2B 8E
8F48- 86 FF 85 19 84 1A 20 62
8F50- 8B A6 F5 86 08 A2 19 A0
8F58- FC 20 38 8B C6 08 F0 28
8F60- A4 08 C4 1D F0 EF A5 19
8F68- A4 1A 20 F9 EA A5 F6 A4
8F70- F7 20 7F E9 A5 9D F0 06
8F78- A5 A2 45 FF 85 A2 A6 19
8F80- A4 1A 20 2B EB 4C 55 8F
8F88- 60 A0 02 84 9F C6 9F 30
8F90- F7 A5 9F A8 45 FF 6A B1
8F98- FA F0 F2 20 62 8B 90 07
8FA0- A8 A9 FA A2 00 F0 05 AA
8FA8- A9 F8 A0 00 48 20 75 8B
8FB0- 68 AA B5 00 85 CE B5 01
8FB8- 85 CF A6 FF F0 0D B5 CE
8FC0- 48 B5 1E 95 CE 68 95 1E
8FC8- CA 10 F3 A4 F5 88 84 9D
8FD0- A0 FC A2 1E 20 38 8B A2
8FD8- CE 20 38 8B A0 04 B1 1E
8FE0- 48 B1 CE C0 01 D0 02 49
8FE8- 80 91 1E 68 91 CE 88 10
8FF0- ED C6 9D D0 DB F0 96 FF
8FF8- FF FF FF FF FF FF FF FF
```

MICRO

MICRO Software Catalog

Name: **CONST**
System: North Star and Apple
Memory: 32K minimum
Language: BASIC
Description: This program was written to do quantity and sizing take-offs for residential and small commercial structures. To operate the program, the user has only to answer questions concerning room sizes and type of construction.
Price: $75.00; listing $60. Includes diskette, on-line documentation, support.
Author: David Lovejoy
Available: Computing Interface
1918 Carnegie Lane #C
Redondo Beach, CA 90278

Name: **BITPAK (for teachers)**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: DOS 3.3/3.2; printer option
Description: Consists of super decimals, long division with remainders, and Super Etch-A-Sketch. The first two are serious CAI programs for grades 1-9. Will do operations with decimals or whole numbers, and long division with remainders. You select the size of the numbers, not levels. Grades work, has traps, and field tested. The third program will sketch designs on the Lo-Res screen with save-to-disk option. Keyboard version, change colors, erase, X-Y coordinates displayed.
Price: $24.00
Author: [IA]:Calabrese
Available: Bit'N Pieces Series
P.O. Box 7035
Eire, PA 16510

Name: **Net-Works**
System: Apple II or Apple II Plus
Memory: 48K
Hardware: 1 drive, DOS 3.3, Applesoft in ROM, D.C. Hayes Micromodem
Description: Bulletin board and computerized message system for Apple II. Features speedy log-on, electronic mail with security provisions, downloading programs, editing, much more. May be used in conjunction with Computer Station's "Auto Modem" for establishing a business communication network, office to home message system, bulletin board service, etc. System operator has complete control of who uses the system.
Price: $124.95 includes disk plus full documentation in sturdy 3-ring binder
Available: Computer Station
11610 Page Service Dr.
St. Louis MO 63141
(314) 432-7019

Name: **The Normalcy Life Dynamic**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II Plus, Disk II
Description: Do you think you want to be "normal"? Perhaps you don't! You'll see, after playing the games on this disk. In fact, your whole perception of normality may change. This disk includes such games as *The Mine Fields of Normalcy*, *Depth Charge!*, *Mystery Code*, and *Deep Sea Treasure*, totally unique games that challenge beliefs as well as skills. Some of the best sound effects ever heard, Hi-Res.
Price: $15.95 includes disk, game card
Available: Avant-Garde Creations
P.O. Box 30161
Dept. MCC
Eugene, OR 97403

Name: **Applied Educational Systems Grade Reporting System**
System: Apple II Plus using Microsoft Softcard
Memory: 48K
Language: Microsoft BASIC-80 with CP/M operating system
Hardware: Apple II, Radio Shack Models 1, II, and III and PET
Description: The *AES Grade Reporting System* produces professional, full-sized report cards for schools of up to 2500 students, and provides the following summary reports: honor rolls; rank in class listings; GPA listing; summary attendance list; class lists; failure and incomplete list; frequency distribution of grades by teacher, course, student year, and department; homeroom lists; permanent record labels; and mailing labels. The system is menu driven and uses a low-cost, automatic mark sense card reader for data entry.
Price: $2000 includes installation, one-day training session, operating manual
Author: Robert C. Hamilton
Available: Applied Educational Systems
RFD 2, Box 213
Dunbarton, NH 03301

Name: **VISI-CAIDS**
System: Apple II
Memory: 32K
Language: Applesoft
Hardware: Printer, 1 or 2 disk drives
Description: *VISI-CAIDS* is a companion package of formatting aids for use with VisiCalc™ text files. The "Label Splitter" creates a new text file, compatible with VisiCalc™, which divides wide label entries in a selected column into two or more narrower columns. The "Width Adjuster" prints VisiCalc™ data into variable width columns and can simulate a split screen on the printer. Also includes "Formula Reader" with special features and a "Print File Reader."
Price: $34.95 includes one disk plus instructions
Author: Charles Harrison
Available: Data Security Concepts
P.O. Box 31044
DesPeres, MO 63131
(314) 965-5044

Name: **Space Adventure Pak**
System: Atari 400/800
Memory: 16K cassette
24K disk
Language: BASIC
Hardware: Joysticks and paddles
Description: *Space Adventure* is an arcade game package that includes two action graphics and sound programs for the Atari. "Space Wars" is a high speed space battle between you and your Atari. "Shootout" is a cannon fight between you and your opponent. Uses player/missile graphics ability of your Atari computer system. Other software is available. Write for complete list.
Price: $12.95 cassette ppd
$15.95 disk ppd
Author: Russell A. Grokett, Jr.
Available: Kinetic Designs
401 Monument Rd. #171
Jacksonville, FL 32211

Name: **CHAT**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft in ROM or Language Card
Hardware: Hayes Micromodem, Disk II, printer optional

Software Catalog *(continued)*

Description: *CHAT* is the communications package that offers you the freedom and fun of simple conversation. Save incoming data in a large 26K buffer. Edit, print, or store on disk what you save. Use the input anything line editor to create text files. Transfer or receive text files or BASIC programs to or from other computers. Features are: automatic log-on to networks, simple configuring to your system, character filter, non-keyboard characters, answering the phone — all clearly explained.

Price: $40.00 includes manual, four Applesoft programs, three text files and *CHAT* binary code on a diskette, DOS 3.3
Author: Robert W. Lovell
Available: Lovell's
4205 Biltmore
Corpus Christi, TX
78413
(512) 852-3096

Name: **OSI BASIC Enhancer**
System: OSI C1P/Superboard/C4P
Memory: 8K
Language: Machine code w/BASIC-IN-ROM
Hardware: C1P, Superboard, C4P

Description: BASIC programmers who want real power over their awkward stock system will love this one. Get real delete action, replace cursor with one of your own choice (defaults to checkerboard square), commands to RENUMBER your programs to make them easy to read, AUTOSEQUENCER will save you from typing in line numbers, screen control to stop scrolling I key to running BASIC. LOAD and SAVE files with filenames on a token I/O system to reduce load-save times by 50%. Runs in approximately 1.5K of RAM. Send $1.00 for complete catalog.

Price: $19.95 postpaid includes autoload, autorun cassette only, User's Manual and bug-free guarantee
Author: Timothy W. Jackson
Available: Computer Science Engineering
57 Beals St. Rm. 57-12
Brookline, MA 02146

Name: **Notewriter**
System: Apple II Plus
Memory: 48K
Language: Assembly, Applesoft
Hardware: Soundchaser 3 Voice Synthesizer Card, Soundchaser Music Keyboard and Interface Card

Description: *Notewriter* is a unique program that transcribes music played live on the Music Keyboard to the monitor screen in real time. The score can then be edited in its entirety and printed out on a graphics printer. A click track is used to sync the music entry with the music notation to give accurate rhythmic representation.

Price: $100.00 includes software and documentation
Available: Passport Designs, Inc.
785 Main Street, Suite E
Half Moon Bay, CA
94019

Name: **Falcons**
System: Apple II and Apple III
Memory: 48K
Language: BASIC
Hardware: Apple II, Apple II PLus, Apple III

Description: Invaders-style game with five levels of invading forces to be repelled. Very challenging and fast-paced. Succeeding games, if you get through a complete game, are more difficult.

Softwara Catalog *(continued)*

Price: $29.95
Author: Eric Varsanyi and Thomas Ball
Available: Piccadilly Software Inc.
89 Summit Ave.
Summit, NJ 07901
(201) 277-1020

Name: **Kamakaze Education Pack**
System: OSI C1P
Memory: 8K
Language: BASIC
Description: Four educational programs in one. Send your men on a tank destroying mission by correctly answering a question from Spelling Drill, Addition Drill, Multiplication Drill, or Place Value Drill.
Price: $15.00
Author: Henry Svec
Available: Henry Svec
668 Sherene Terrace
London, Ontario
Canada N6H 3K1

Name: **Type**
System: SDOS or SDOS/MT
Memory: 48K minimum
Hardware: 6800/6809 CPU with CRT, disk and printer

Description: *Type* is a document-formatting program, used in word processing or document production. Commands embedded in raw text files processed by TYPE control the formatting of that text on the output device. Output formatting includes full justification, page width and depth, page numbering, centering, spacing, titles and table of contents generation. *Type* is used in conjunction with the SD Screen Editor for easy data entry.
Price: $140.00 includes Type Program, 100-page manual
Author: AMS
Available: Software Dynamics, (exclusively)
211 W. Crescent Suite G
Anaheim, CA 92801
(714) 635-4761

Name: **Soft Pretzels for OSI**
System: All OSI cassette-based systems
Memory: 8K
Language: OSI BASIC
Hardware: Standard systems (cassette only!)
Description: High speed, real time, action arcade games including "Humanoid Defender," "Lunar Rescuer," "Rebel Gunner," "Spiderfood," "Rampaging Robots" and many more! All include color and sound at no extra cost. Sound for C1P is fully supported! Instruction sheet for installing sound on C1P available. Educational programs also available.
Price: $1.00 for photo-illustrated catalog, includes $1.50 credit on first order. Color and sound for C4Ps, sound for *all* C1Ps.
Author: Bob Retelle, et. al.
Available: Pretzelland Software
2005 Whittaker Rd.
Ypsilanti, MI 48197

MICRO

MICRO Hardware Catalog

Name: **RS-232 DB-25 Pin Reconfiguration Adapter**
Description: Printed circuit card with one male and one female DB25 connector mounted on it and incorporating a special "matrix switch." This device enables users to instantly mate almost any serial I/O device to any computer by rerouting RS-232C signals.
Price: $59.95
Available: Mountain Computer Inc.
300 El Pueblo Rd.
Scotts Valley, CA 95066
(or local Apple dealers)

Name: **16K RamBoard**
System: Apple II or Apple II Plus
Memory: 16K Dynamic RamBoard
Description: A 16K RamBoard that expands an Apple II or Apple II Plus 48K to 64K. The RamBoard is compatible with all Apple II languages and software. It enhances operations by allowing larger languages, data bases and programs. It also greatly improves the capability of CP/M, Pascal, Fortran and Cobol.
Price: $129.95 introductory offer includes RamBoard and complete installation and operating instructions
Available: ConComp Industries
8338 Center Drive
La Mesa, CA 92041
(714) 464-8715

Name: **MFD Mini-Disk Systems**
System: AIM 65, KIM, SYM
Language: DOS supports AIM monitor, Assembler, BASIC and PL/65
Hardware: Controller, Drive(s), Cable
Description: Mini-Disk storage system including DOS, drive controller, cable and user's manual for AIM 65, KIM and SYM computers. Controller available for either the AIM expansion bus or for the SS-50 bus. AIM-to-SS-50 motherboard adapter available.
Price: Mini-Disk Systems from $599.95, Adapter (M-65/50) is $89.95.
Available: Percom Data Co., Inc.
11220 Pagemill Rd.
Dallas, TX 75243
(214) 340-7081
(and authorized Percom dealers)

Name: **The DOS Switch**
System: Apple II, Apple II Plus
Description: Allows a DOS 3.3 equipped Apple II system to boot DOS 3.2 or DOS 3.3 diskettes simply by flipping The DOS Switch. You can conveniently use your valuable copy-protected/unMUFFINable DOS 3.2 software, without the BASICs diskette. Easy to install and use. Two models: DS-1 (uses your P5 and P5A PROMs), and DS-2 (3.2 boot PROM installed).
Price: Model DS-1 $29.95
Model DS-2 $44.95
Available: Computer Micro Works, Inc.
P.O. Box 33651
Dayton, Ohio 45433
(or Apple dealers)

Name: **Dithertizer II**
System: Apple II
Memory: 48K
Language: Applesoft and Assembler
Hardware: Board
Description: Package consists of board, Sanyo VC 1610X camera, and cables. Also included is software for image contouring. The Dithertizer converts input into dithered images which produce the appearance of gray scales on the Apple II screen. Pictures may be saved to disk and the number of scan levels may be increased.
Price: $656.00 includes S&H
Available: Peripherals Plus
39 E. Hanover Ave.
Morris Plains, NJ 07950

Name: **GMS 6519 Floppy/Printer Controller**
System: 6500/6800
Hardware: 6" × 9.75" module
Description: Controls two 5¼" floppy disk drives and a printer, with eight programmable I/O lines, 1 MHz or 2 MHz operation, base address and enable/disable switches, over voltage and reverse polarity protection. Optional 4K operating system, optional 6K 6502 assembler, both compatible with System 6T. Can drive floppies such as Shugart, Teac, Pertec; printer such as Centronics.
Price: $246.00, single piece qty.
Available: General Micro Systems
1320 Chaffey Ct.
Ontario, CA 91762
(714) 621-7532

Name: **Andromeda ROM Board**
System: Apple II
Memory: Any
Description: The Andromeda ROM board permits you to plug many utility programs into your Apple II and access them instantly without loading them from disk. You can install 2K PROMS, 4K PROMS, or even 2K RAM chips in each of 2 memory sockets. Comes with a utility ROM with five built-in options to apply to your Applesoft programs: automatic line numbering, list control, DOS expunge, alphabetize disk catalog, and restore a crashed program. Many more PROMS are available.
Price: $125.00
Available: Computer Data Services
P.O. Box 696
Amherst, NH 03031
(603) 673-7375

Name: **The Findex Microcomputer**
Memory: BASIC configuration is 80K of RAM, expandable to 2 million characters
Language: Supports Business BASIC, Cobol, Fortran, Pascal, APL, PL-1
Description: Combines memory, expanded keyboard, gas plasma display, disk drive and printer in a package no larger than most electric typewriters. This 31-lb. microcomputer system interfaces to a variety of outside peripherals, like larger printers or multiple hard disk drives. Also available in applications software packages for business, medical offices, etc.
Price: $6,980 - $20,000, depending on peripherals
Available: Findex
20775 S. Western Ave.
Torrance, CA 90501

Name: **ROM Simulator**
System: Apple II Development System
Memory: 2K
Hardware: Double-sided board
Description: Double-sided, gold-plated board for developing software from host computer (Apple) to target computer (usually SUPERKIM in Lamar Instruments Develepent System) to be placed in the ROM. Reduces time required to program. Can be used to increase RAM available in Apple.
Price: $295.00
Available: Lamar Instruments
2107 Artesia Blvd.
Redondo Beach, CA 90278
(213) 374-1673

MICRO

16K Apple II, No I/O Slots: $900.00

Would you even consider purchasing an Apple II that cannot be expanded, even though you would save $200.00? Of course not! An Apple that can't be expanded is worthless. So why would anyone want to buy a lower case adapter that can't be expanded to meet the changing needs of all Apple owners?

Rather than design our product as one monolithic item that you buy once and pray it doesn't become obsoleted, Lazer MicroSystems offers a modular system with power to grow; a complete set of integrated components that can be purchased as needed. You don't need to lay out a large sum of money all at once, nor do you have to pay for unnecessary features to get the features you do want. And you won't have to throw anything away should you decide to upgrade tomorrow. The choice is yours: do you want to take a chance with a system that costs a few dollars less, or do you want to purchase the state-of-the-art system that can be expanded to meet all of tomorrow's needs?

Start with our field-proven Lower Case +Plus. At $64.95 it offers twice the features found on any other lower case adapter. Software on diskette, inverse-only mode, two on-board character sets (word processing and graphics), excellent documentation, and our exclusive "expansion socket" -- your key to the future. Not all lower case adapters were created equal, ours was created better!

GRAPHICS & LOWER CASE CHARACTER GENERATOR
FOR THE APPLE II COMPUTER

Some of the features of our system include:

1) Our system is completely compatable with any piece of software that will work with the "Dan Paymar Lower Case Adapter."

2) Our system works with every word processor we've seen for the Apple II including EasyWriter and unmodified Applewriters. In fact, Lazer's lower case system works with more word processors than anyone else's

3) Our hardware works fine with all printers and 80-column boards.

4) You CAN order a Lower Case ± Plus specifically prepared for your Apple.

5) We offer a one-year limited warranty on our board.

6) Our board is easily installed. Most beginners can install it within twenty minutes.

7) The lower case system is fully transparent to your Apple. The Lazer system does not use up any of the Apple's available RAM, nor does it slow the operation of your computer down in any way. In fact in some instances our equipment speeds up existing programs!

8) Lazer MicroSystems' hardware products were designed by Randy Hyde. Randy, as a hardware and software engineer, has been designing products for the Apple II since 1977. With four years of experience behind our products, you can be well assured that Lazer's components are well integrated into the Apple system and you will encounter few difficulties using the Lazer lower case system with any software available for the Apple II.

9) Lazer MicroSystems, Inc., is a real live corporation operating out of real offices in Southern California. Most manufacturers of lower case equipment operate out of their bedroom. Who do you think will be around in two years to support their product?

& SHIFT KEY UPPER/LOWER CASE CONTROL

Next add our Keyboard +Plus. The Keyboard +Plus turns your Apple's uppercase only keyboard into a professional 128-ASCII character keyboard. The Keyboard +Plus also features a 64-byte type-ahead buffer that drastically improves the performance of existing programs. lazer's buffer can be cleared or disabled if desired.

MORE PRODUCTS FROM LAZER

Character Set +Plus: Add two more ROM-based character sets to your Lower Case +Plus! With the CS+ you have a total of four character sets on-board. Choose from any of the 25 fonts provided on the Lower Case +Plus disk, or create your own. Only $24.95.

Double Vision +Plus: At last! Good lower case display for Computer Stop's Double Vision Board. In addition to the improved display, our DV+ gives you two on-board character set (std and graphics) as well as our famous expansion socket that lets you add our Character Set +Plus and Graphics +Plus peripherals. Only $39.95.

Lower Case +Plus II: for the budget minded Apple owner. A low-cost chip that replaces the character generator ROM on revision seven and later Apple motherboards. For $29.95 you get the Lower Case +Plus II, Pascal and BASIC software on diskette, and complete installation instructions. The Lower Case +Plus II is not expandable like our Lower Case +Plus, but we do offer a $20.00 trade-in allowance if you decide to upgrade to a Lower Case +Plus in the Future.

Lazer
MICRO SYSTEMS INC.
1791-G Capital
Corona, CA 91720
(714)735-1041

Ordering Information:

Lazer MicroSystems' products are available from reputable distributors, dealers, and mail order houses worldwide. If your local dealer cannot demonstrate our products for you, tell him he can get them from Computer Data Services, Micro-D, Softsel, Lazer MicroSystems, and other distributors. If your dealer refuses to stock Lazer products, give Lazer, Huntington Computing, or Consumer Computers a call--the product is always available off the shelf.

Before you purchase anybody's lower case adapter, ask for a copy of our free booklet "Keyboard Enhancers/Buffers and Lower Case Adapters: From the Inside Out", it will explain all the advantages and disadvantages of installing lower case equipment in your Apple II. Or give us a call--we'd be more than happy to discuss our product line with you.

Dan Paymar Lower Case Adapter TM Dan Paymar.
Apple...TM Apple Computer Inc.
Double Vision TM Computer Stop.
+Plus TM Lazer MicroSystems.
Note: EasyWriter and unmodified AppleWriter usage requires optional ROM

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

6502 Bibliography: Part XXXIX

1114. From the Core (March, 1981)

Barden, Frank, "Super-Text Patches for the MX-80," pg. 4.
Combine the best qualities of this word processor with the versatile MX-80 and avoid having the control codes of each interfere with the other.

Whittaker, Alec, "As BASIC as Possible," pg. 7.
A short tutorial on fitting some simple routines together.

1115. SoftSide 3, No. 6 (March, 1981)

Patton, Garry J., "Screen Capture," pg. 19.
A short routine for the Atari (with Apple translation) to print the screen to the printer.

Voskuil, Jon, "Math Decathlon," pg. 21-25.
An Applesoft listing of a game/tutorial in four parts.

Pelczarski, Mark, "Developing Data Base," pg. 34-35, 57.
Part 7 and the final chapter in this series on a Database for the Apple and Atari.

Morris, W. and Cope, J., "Flags," pg. 45-46.
An Atari program to display national flags on the screen.

Steenson, David, "Strategy Strike," pg. 52-57.
A game for the Apple II or TRS-80.

Truckenbrod, Joan, "Three-Dimensional Rotation," pg. 66-67.
A computer graphics tutorial for 6502 micros.

Voth, Mitch, "Mini Golf," pg. 68-70.
A game for the Apple.

Various "Apple One Liners," pg. 70.
Several one-line programs for the Apple.

Johnson, Paul, "Atari Memory Upgrade," pg. 92-93.
A hardware article on easy 8K to 16K conversion for the CX852 memory module in the Atari 800.

1116. 73 Magazine No. 247 (April, 1981)

Jensen, Carl, "Apples: Five Bits or Eight," pg. 66.
Ham operation on 5-bit RTTY (Baudot) or 8-bit RTTY (ASCII) with the Apple II; hardware and software.

1117. Compute! 3, No. 3, Issue 10 (March, 1981)

Mansfield, Richard, "Taking the Plunge — Machine Language Programming for Beginners," pg. 20-25.
A tutorial for PET users.

DeJong, Marvin L., "Computer Communications Experiment," pg. 28-33.
Hardware and machine language routine for a 6502-to-RS-232C to Modem interface.

Sander, Louis F., "Getting the Most from Your PET Cassette Deck," pg. 42-46.
How to insure the reliability of your cassette for the PET.

Albrecht, Bob and Firedrake, George, "The Mysterious and Unpredictable RND," pg. 4852.
Part 3 of several installments for PET users.

Oakes, Peter, "A CAI Program Called Linear Equation," pg. 54-59.
A computer assisted instruction (CAI) math program for the PET.

Clarke, Jack, "HEX Conversion," pg. 60-61.
Using the 6502's Decimal Mode — a tutorial.

Ostrowsky, Sherm, "Clearing the Apple II Low-Resolution Graphics Screen," pg. 6266.
Fast screen clear techniques for the Apple.

Mauney, Gene A., "Fun with Apple and Pascal," pg. 68-70.
Help for the user of Apple/Pascal.

Patchett, Craig, "Designing Your Own Atari Character Sets," pg. 72-77.
A tutorial for Atari users.

Gropper, D.M., "Atari BASIC — A Line Renumbering Utility," pg. 78.
A short program for the Atari.

Baker, Robert W., "Atari Memory Dump and Disassembler," pg. 80-84.
A handy utility program which allows you to examine RAM or ROM memory with either a dump or disassembly listing.

Wrobel, Joseph J., "Formatted Output for Atari BASIC," pg. 84-85.
A utility to simulate 'print using' and TAB functions on the Atari.

Howell, R.A., "Random Color Switching While Idle," pg. 85-86.
A routine for the Atari.

Berger, Tom R., "A Small Operating System: OS65D The Kernel," pg. 87-88.
Part 3 of three installments on an operating system for OSI micros.

Stanford, Charles L., "A Six-Gun Shootout Game for the OSI C1P," pg. 88-91.
A game for the OSI micros.

Brandon, Eric, "Keyprint Revisited," pg. 92-94.
Here is a KEYPRINT routine for 'old ROM' PETs.

Butterfield, Jim, "Learning About Garbage Collection," pg. 96-100.
Reclaiming memory on the PET.

Fremont, David Malmberg, "PET Machine Language Graphics," pg. 102-110.
A tutorial on PET graphics.

Campbell, Gordon, "PET Exec Hello," pg. 124-128.
A trilogy of small programs which allows you to use a 'Hello' file on disk.

Busdiecker, Roy, "Universal Tape Append for PET/CBM," pg. 132-137.
How to combine programs on CBM systems.

Clements, William C., Jr., "A Vocal Hex Dump for the KIM-1," pg. 146-149.
A program for the KIM-1 and an inexpensive speech synthesis unit.

Chamberlin, Hal, "Expanding the KIM-Style 6502 Single Board Computers: The Modified KIM Bus," pg. 150.
Part 3 of three articles on a modified bus.

1118. Interface Age 6, Issue 4 (April, 1981)

Moore, Herb, "Sounds of the Atari — In BASIC," pg. 50-54.
Part 2 on color graphics for the Atari.

Edelson, Roger H., "The Microsoft Z-80 SoftCard — CP/M for AP," pg. 60-63.
A review of a card to expand the capabilities of the Apple.

1119. Interactive Issue 4 (March, 1981)

Caldwell, Sam, "Communications for the Handicapped," pg. 3-7.
Use of the AIM 65 with some minimal hardware and a simple basic program to assist the non-verbal handicapped.

McIntosh, Bruce, "Assembly Offset," pg. 10-11.
How to make the AIM 65 assembler offset object code for EPROMS.

April, Georges, Emile, "AIM 65 Assembler OUTPUT Formatter," pg. 12-13.
Output formatter and Centronics printer driver.

Corder, Mike, "Solid Graph Plot," pg. 17-20.
BASIC program and machine code for an AIM 65 plot routine.

Shafer, Marvin D., "Improved Plot Routine," pg. 21-22.
A utility for the AIM 65.

April, Georges, Emile, "AIM 65 Monitor Bit Patterns," pg. 24.
A table of use to AIM 65 users.

1120. PEEK (65) 2, No. 3 (March, 1981)

Jones, David A., "Cassette Corner," pg. 4-5.
A cassette loader with checksum which is quite a bit shorter than the OSI autoload.

Morris, E.D., Jr., "Improve Your Garbage Collection," pg. 6-7.
Memory recovery for the OSI system; a permanent hardware solution to the problem.

Holt, Ken, "Assembler Programmer's Guide to OSI Board Interfacing," pg. 12, 9-11.
$30 board addressing, OSI 470 board, 470 board addressing, OSI 500 board, 500 board addressing, OSI 510 board, 510 board addressing.

1121. KB Microcomputing 5, No. 4, Issue 52 (April, 1981)

Baker, Robert W., "PET-Pourri," pg. 10-13.
News on the VIC-20 Commodore color computer, new PET software, etc.

Messent, Andrew N., "Save It with CASSY," pg. 38-41.
A versatile machine-language program for the OSI C1P which saves any area of memory to the cassette recorder.

Jones, Edward C., "Energize Those OSI Peripheral Ports," pg. 80-82.
Hardware modification for the Challenger 1P or OSI Superboard II to provide for a printer or modem.

Peterson, Craig, "Apple Sector Counter Extraordinaire," pg. 113-114.
An Apple utility to keep track of the free, unused sectors on the floppy disk.

Blank, Jerry, "Hidden Features Exposed," pg. 156-157.
Some unexpected capabilities of the Atari microcomputers, including a hardware mod for upgrading the Atari 400 to 16K.

Bradshaw, Cyril, "OSI Clear-Screen Command," pg. 172-173.
A disk modification which blanks the OSI screen in milli-seconds.

Hansen, Jim, "A Tale of Two Screen Dumps," pg. 174-176.
A utility for the Apple/Paper Tiger combination.

1122. The Cider Press (January/February, 1981)

Staff, "The Best of Cider Press, 1980."
Over 100 pages of the best articles published in the Cider Press in 1980.

1123. L.A.U.G.H.S. 3, No. 3 (March, 1981)

Roe, David, "There is a Better Way — Machine Language Portion," pg. 3-4.
Routines to locate screen array, store or display Hi-Res screen in that array, and to switch graphics on or off in the Apple.

1124. The Apple-Dillo (March, 1981)

Teas, George, "Pascal Primer," pg. 6-7.
Notes on creating a UNIT and including it in the SYSTEM.LIBRARY.

1125. The Cider Press (March, 1981)

Hartley, Tim, "Changing Diskette Colume Numbers," pg. 4.
An aid for Apple disk users.

Norris, Paul, "Pascal and the F8 Monitor ROM," pg. 11.
Tips for the Applc Pascal user on accessing certain Apple ROM functions.

Sugou, Shin'Ichirou, "Pascal Assembly — An Example Program."
Utility routines for Apple Pascal users.

1126. Byte 6, No. 4 (April, 1981)

Minton, R.B., "A Graphic Execution Display," pg. 34.
A program for the OSI Superboard to display graphically how fast a program is running.

Carlson, Edward H., "Challenger Writes on Comprint," pg. 310-312.
A 6502 assembly language program that interfaces a 6522 VIA parallel port on the OSI Challenger II with a COMPRINT 912 printer.

1127. IEEE Transactions On Education, Vol. E-24, No. 1 (February, 1981)

Hanson, Donald F., "A Microprocessor Laboratory for Electrical Engineering Seniors," pg. 8-14.
Instructional Apparatus including the AIM 65 microcomputer and an auxilliary educational interfacing unit is described.

1128. Proc, 1981 Southeastern Section Annual Meeting, Amer. Soc. for Eng. Education, (April 5-7, 1981)

Hanson, Donald F., "A Microprocessor Laboratory Course Based on an AIM 65 Solderless Interfacing Unit," pg. 1-7.
A 6502-based educational tool for Electrical Engineering students.

1129. Sym-Physis 2, No. 1, Issue 7 (Jan/Feb/Mar, 1981)

Anon., "A Bell for the KTM-2 and/or KTM-2/80," pg. 4.
A bell function for the SYM-1.

Anon., "Misc notes for SYM-1," pg. 4-6.
Controlling I/O from BASIC; cassette problem and fix; wiggle your chips and flex your boards; the RAE user function; disk systems, etc.

Anon., "A Deductive Story — Part I."
All about BASIC and BAS-1 for the SYM-1.

Newman, John, "A Kansas City Standard Tape Dump," pg. 9-16.
A utility for the SYM-1.

Anon., "A Wide Screen Hex/ASCII Memory Dump," pg. 22-25.
An easily relocatable memory dump utility for the SYM-1.

1130. SoftSide 3, No. 4 (January, 1981)

Pelczarski, Mark, "Developing Data Base — Part Five," pg. 16-18.
More on the search routine for this utility for the Atari, the Apple II and the TRS-80.

Edmunds, William, "Computer Space," pg. 40-41.
A game for the Apple.

Morris, William, "Fugue," pg. 74-76.
A music program for the Atari.

Truckenbrod, Joan, "Computer Graphics," pg. 83-84.
Tutorial with demo example for Apple Hi-Res.

1131. Abacus II 3, Issue 2 (February, 1981)

Freeman, Larry, "DOS Type," pg. 2-5.
A routine to identify the type of DOS and disk type active in your Apple.

Zirak, Victor, "ASCII Memory Dump," pg. 8.
View the contents of the Apple's RAM memory with this routine.

Davis, James P., "M.E.C.A. — An Expanded Menu," pg. 9.
A menu program for the Apple disk.

Yee, David R., "Long Division," pg. 16.
An extended precision long division routine for the Apple.

1132. The G.R.A.P.E. Vine 2, No. 2 (March, 1981)

Lawson, Steve, "G.R.A.P.E. Font Converter," pg. 7.
A program for the Apple to convert a standard size font into a large size font.

Ude, Art, "Conversion of Biblical Measures," pg. 8-9.
Cubits to metric units.

1133. Stems from A.P.P.L.E. 4, Issue 3 (March, 1981)

Wittke, Walton L., "Look," pg. 4.
An Applewriter utility that lists the catalog and lists the first 12 lines of a designated text file.

Ward, Dennis, "DOS 3.3 Tool Kit Assembler Hints," pg. 7.
How to get the most out of the Tool Kit assembler on the Apple.

Ward, Dennis, "Professional Hi-Res Made Easy," pg. 7.
Some hints for using the Tool Kit to modify an Apple graphics page.

1134. The Michigan Apple-Gram 3, No. 3 (March, 1981)

Anon., "Apple II Mini-Assembler F666G," pg. 12-13.
A short tutorial on the Apple monitor's mini-assembler.

Anon., "IAC Apnote: Tabbing with Apple Peripherals," pg. 13-14.
This driver allows the user to tab normally without substituting POKE 36,X for TAB(x).

Zager, Bill, "Apple Concordance Revisited," pg. 27-28.
An Apple utility to print out a list of a BASIC program's variables.

1135. The Harvest 2, No. 8 (April, 1981)

Andersen, Chuck, "Diskette Nibblizing," pg. 11-12.
A tutorial on how data is stored on the Apple disk.

Thompson, C.J., "NIFFUM — A DOS 3.3 to 3.2 Converter," pg. 19-20.
A utility for the Apple disk system.

1136. Washington Apple Pi 3, No. 3 (March, 1981)

Neuman, David, "CP/M for the Apple," pg. 10-16.
A general description of the CP/M for Apple.

Decker, R.J., "More Tips of the MX-80," pg. 22.
Notes on horizontal tabbing on the Epson MX-80 printer.

1137. Personal Computing 5, No. 4 (April, 1981)

Gilder, Jules H., "How to Boot Binary Programs," pg. 56-57.
Use a binary program as your Apple disk Hello.

1138. Creative Computing 7, No. 4 (April, 1981)

Parr, James, "Apple as Time-Sharing User," pg. 60-65.
A helping hand into the world of time-sharing for Apple users.

Jacobs, Jake, "Landing Simulator," pg. 156-166.
Using the Apple as a landing simulator for aircraft.

1139. Radio Electronics 52, No. 5 (May, 1981)

Hyypia, Jorma, "Learning about Microprocessors," pg. 45-48.
Discussion of the SYM-1 as a learning aid.

Gupton, James A., Jr., "Computer Control for the Unicorn-1 Robot," pg. 53-55.
The use of the KIM-1 and SYM-1 for robot control is discussed.

1140. MICRO, No. 35 (April, 1981)

Rhodes, Ned W., "S-C Assembler Modifications," pg. 7-10.
Adding back the multiply routine; automatic line numbering; etc.

Kovacs, Bob, "Memsearch for the AIM 65," pg. 17-20.
A machine language routine which can scan memory for a user-specified sequence. Includes a wild-card feature.

Orton, Ralph, "SYM Time-Remaining Timer," pg. 37-39.
This SYM program measures elapsed time and sounds an alarm.

Campbell, Gordon A., "Oh No — It's Garbage Collect!", pg. 43.
All about garbage collection (memory management) on the 6502 and how to avoid those long delays.

Sogge, Glenn R., "Integer BASIC Internals," pg. 65-66.
A sorted list of Apple Integer BASIC memory locations and routines.

1141. AppleGram 3, No. 3 (March, 1981)

Sander-Cederlof, Bob, "Cryptogram Solver," pg. 8-10.
A short routine for the Apple to help with decoding those cryptograms in the Sunday paper.

Wiggington, Randy, "Fast Garbage Collection," pg. 21-25.
A rapid garbage collector program which reconstructs string memory, rapidly cutting down the length of those times when your Apple program just seems to die.

Sander-Cederlof, Bob, "Modify DOS Commands," pg. 29-30.
A utility for the Apple which changes the DOS commands and updates the DOS on your disk.

1142. The C.I.D.E.R. Press 3, No. 2 (March/April, 1981)

Lingwood, David A., "Word Processor Evaluation Guide," pg. 3-6.
A review of features to look for in selecting a word processor for the Apple.

Greene, Amos, "Catalog Interrupt," pg. 9.
A utility for the Apple which permits listing only a portion of the catalog.

1143. Softalk 1, No. 8 (April, 1981)

Wagner, Roger, "Assembly Lines," pg. 25-28.
The latest installment in this continuing series deals with the various addressing modes for the 6502 microprocessor.

1144. The Michigan Apple-Gram 3, No. 3 (April, 1981)

Tuttleman, Roger, "Text File Reader," pg. 2.
An Apple utility to look at sequential text files.

Zager, Bill, "Printing Mail Power Data on Two-Column Labels," pg. 4-5.
Printing labels two across using the Apple.

1145. Dr. Dobb's Journal 6, Issue 4, No. 54 (April, 1981)

Harris, David C., "Important Features of the PCNET Protocol," pg. 47-52.
A protocol for data communications is being developed so that micros such as the PET and Apple can communicate with most mainframe computers using a universal system.

1146. Call —A.P.P.L.E. 4, No. 3 (March/April, 1981)

Rosing, Mike and McLauren, Keith, "Pascal Internals," pg. 9-21.
A primer in several parts: booting process; I/O routines; Pascal directory; 6502 machine code and p-code; and two appendices on location of machine registers and source code for a disassembler.

Manly, Ken, "Keeping Up to Date," pg. 49-54.
An Apple utility to make EXEC file backups of BASIC programs in development.

Hendel, David, "ROM Mover," pg. 53-54.
An Apple utility for moving the F8 ROM to the D0 socket.

Golding, Val J., "Data Statement Writer," pg. 58.
A utility for the Apple.

Hartley, Tim, "Programs to Modify VTOC (DOS3.3)," pg. 63.
Routines to show tracks 30-34 in use and to show tracks 30-34 as completely free.

1147. Applesauce Vol. 2, No. 3 (January/February, 1981)

Jordan, Tricia, "Easy or Pie," pg. 3-8.
Experiences with two Apple word-processing systems.

Hyde, Randy, "LISA's Internal Structure and Customization," pg. 10-12.
How to modify the LISA assembler so that special purpose functions may be included.

Mazur, Jeff, "Coping with Apple's High Speed Serial Card: Part I," pg. 24.
Adding handshaking to the High Speed card.

1148. The Seed 3, No. 4 (April, 1981)

Wheeler, Steve, "Ted II Modifications," pg. 8-10.
Add four new pseudo-operations to the Ted II Editor/Assembler.

1149. Apple-Dayton Newsletter 2, No. 4 (April, 1981)

Fox, Dan, "Flies in the Ointment — Bug Found in Apple Renumber," pg. 8.
A problem to watch for in the renumber routine.

1150. Southeastern Software Newsletter Issue No. 25 (April, 1981)

Hartley, Tim, "The Muffin Fix," pg. 7-8.
Modification of VTOC so that Muffin would think that disk tracks 32-34 are in use.

1151. The Apple Peel 3, No. 4 (April, 1981)

Hill, Alan G., "Amper-Reader," pg. 7.
A utility for the Apple.

1152. SoftSide 3, No. 7 (April, 1981)

Truckenbrod, Joan, "Computer Graphics," pg. 26-28.
Perspective drawing on computers such as the Apple.

Morris, William and Cope, John, "Convoy," pg. 34-39.
A game for the Atari.

Summers, Murray, "Applesoft Chaining," pg. 54-55.
A tutorial for the Apple with 3 demo listings.

Voskuil, Jon, "Math Decathlon," pg. 66-68.
This second article for Apple users adds two more events.

1153. Compute! 3, No. 4, Issue 11 (April, 1981)

Thornhurg, David D., "The Commodore VIC-20: A First Look," pg. 26-33.
All about the new Commodore color computer.

Butterfield, Jim, "How to be a VIC Expert," pg. 34.
Advice to PET owners and a demo listing for the new VIC.

Keck, Rick, "Basically Useful BASIC," pg. 36.
An ascending/descending sort routine for the 6502 micros.

MICRO

Statement of ownership, management, etc., required by the act of Congress of October 23, 1962 of MICRO published monthly at Chelmsford, Massachusetts for November 1981.

The name and address of the publisher is MICRO INK, Inc., 34 Chelmsford Street, Chelmsford, Massachusetts. The Editor/Publisher is Robert M. Tripp of Chelmsford, Massachusetts.

The owner is MICRO INK, Inc., Chelmsford, Massachusetts and the names and addresses of stockholders owning or holding one percent or more of the total amount of stock are: Robert M. Tripp and Donna M. Tripp of Chelmsford, Massachusetts.

The known bondholders, mortgagees and other security holders owning one percent or more of the total amount of bonds, mortgages or other securities are: none.

The average number of copies of each issue of this publication sold or distributed through the mails or otherwise to paid subscribers during the twelve months preceding the date shown above is: 13,360.

I certify that the statements made by me above are correct and complete.

Signed: Robert M. Tripp
Editor/Publisher

Advertiser's Index

Next Month in MICRO

Pascal Faature

- **Elementery Pescel Internels** — An introduction to the internal structure of the P-machine and to some of the concepts which underlie the workings of P-code Pascal implementations.
- **Pescel Textfile Lister** — This utility will improve the appearance of your listings by separating them into pages with headings.
- **RELOC** — Allows the Apple Pascal text editor to be used with DOS 3.3 to more easily edit BASIC text files.
- **Pescel Tutoriel, Pert 3** — Last article in our series to help you understand UCSD and Pascal.

Othar Faatures

Shepe Menipulete Progrem — This Applesoft program lets you create a shape table, delete shapes, add shapes from other tables, or change the maximum number of shapes in a table.

Experimentetion end the TRS-80C Color Computer — A summary of the normal capabilities of the TRS-80 Color Computer, and the unit's I/O capability.

An I/O Expension for the AIM — This article describes a method of adding two more V.I.A.'s to the AIM to triple the capacity of its input/output control hardware.

Scrolling List Progrem — Scroll forwards or backwards through a listing to view any part of a BASIC program without requiring a series of keyed LIST commands.

HUNTINGTON COMPUTING

ONE OF THE WORLD'S LARGEST INVENTORIES

VisiCalc List $200.00 .... now **$149.00**

VisiFile List $250.00 .... now **$199.00**

Horriblescope (Disk) List $15.00 .... now **$3.99**

MISCELLANEOUS

Escape from Arcturus	$29.95 now	$25.39
Basic Mailer	$69.95 now	$59.49
Memory Management II	$49.95 now	$42.49
Castle Wolfenstein	$29.95 now	$25.39
Upper Reaches of Apshai	$19.95 now	$16.89
Bridge Tutor	$39.95 now	$25.19
PASCAL Animation Tools		**$45.99**
Hand Holding BASIC		**$84.99**
UT-100 Emulator		**$45.99**
Space Quark	$29.95 now	$25.39
Beneath Apple DOS (book)	$19.95 now	$14.49
Birth of the Phoenix	$14.95 now	$12.69
Goblins	$27.50 now	$22.29
Painter Power	$39.95 now	$33.89
U.S. Constitution	$29.95 now	$25.29
Merger	$49.95 now	$42.49
Super Stellar Trek	$39.95 now	$33.89
LISA	$79.95 now	$47.89
V-Plot (Yuccipa)	$29.95 now	$25.39
V-Stat	$29.95 now	$25.29
V-Print	$29.95 now	$25.39
Brain Surgeon	$49.95 now	$62.49
Info Master	$150.00 now	$127.49
Waterloo II	$49.95 now	$42.39
Fantysland 2041	$59.95 now	$50.99
Torpedo Terror	$24.95 now	$21.19
Speedstar	$139.95 now	$114.69
Kaves of Karkhan	$49.95 now	$42.29
Dos Boss	$24.00 now	$20.39
Memorex Disks		10 for $24.99
Verbatim Datalife-plain w/hubs		10 for $27.99
Dysans		10 for $69.99
Flipsort Box		$29.80
Scotch Disk Cleaner	$29.95 now	$24.99
Atari®/TRS-80®/Pet®		Write for Information

GAMES

Hi-Res Soccer	$29.95 now	$25.29
Apple®-Oids	$29.95 now	$25.39
Wurst of Huntington Computing		**$19.99**
Gobbler	$24.95 now	$21.19
Ultima	$39.95 now	$33.89
Autobahn	$29.95 now	$25.39
Battle Cruiser Action	$44.95 now	$28.89
Gorgon	$39.95 now	$33.89
Super Stellar Trek	$39.95 now	$23.89
Hellfire Warrior	$39.95 now	$33.99
Gamma Goblins	$29.95 now	$25.29
Mission Asteroid	$19.95 now	$17.99
Wizardry	$49.95 now	$42.49
Star Mines	$29.95 now	$25.29
Warp Factor	$39.95 now	$33.99

WURST OF HUNTINGTON

Read Softalk's review on page 87 of the October issue

$19.99

NEW PROGRAMS

Sneakers	$29.25 now	$25.39
Copts & Robbers	$34.95 now	$29.69
Pot-O-Gold I	$39.95 now	$33.89
Pot-O-Gold II	$39.95 now	$22.89
Ring of Saturn	$39.95 now	$33.89
Streets of the City & Trucker	$24.95 now	$21.19
President Elect	$39.95 now	$23.89
Dis-o-Doc	$72.39 now	$61.49
Med Fly Mania	$29.95 now	$24.29
Hungry Boy	$24.95 now	$21.19
Anova	$74.95 now	$63.79
Solitaire	$29.95 now	$25.39
The Source	$100.00 now	$89.99
Race for Midnight	$29.95 now	$25.39
Hi-Res Secrets	$125.00 now	$106.19
Banners I	$34.95 now	$29.69
Complete Graphics (Tablet)	$119.95 now	$101.89
Special Effects (Tablet)	$69.95 now	$60.79
Inkwell Apple® III	$150.00 now	$130.69
Snack Attack	$29.95 now	$25.39
V-Plot	$29.95 now	$25.29
Disasem/65	$29.95 now	$25.39
The Inspector (Disk or Prom)	$49.95 now	$42.39
The Book	$19.95 now	$11.97
Don I (Book)	$8.95 now	$8.05
Using 6502 Assembly Language	$19.95 now	$16.89
Expandaport	$59.95 now	$50.89
Firebird	$29.95 now	$25.39
Thief	$29.95 now	$25.29
Threshold	$23.97 now	$20.38
Target Planner	$195.00 now	$165.49
Paddle-Adapple	$29.95 now	$26.89
So Cal Research	$14.95 now	$13.45
Word Handler	$250.00 now	$212.49

GAMES

Microsoft Adventure	$29.95 now	$25.29
Wizard and the Princess	$32.95 now	$20.99
Flight Simulator	$34.95 now	$29.49
Odyssey	$29.95 now	$25.39
Sargon II	$34.95 now	$29.69
Space Eggs	$29.95 now	$25.29
Hi-Res Cribbage	$24.95 now	$21.19
Lords of Karma (cass.)	$20.00 now	$16.99
Oh Shoot	now	$19.99
ABM	$24.95 now	$21.19
Computer Conflict	$39.95 now	$35.99
Computer Air Combat	$59.95 now	$52.99
Temple of Apshai	$39.95 now	$33.09
Zork	$39.95 now	$33.99
All Nibble Software		**15% off list**
Robot Wars	$39.95 now	$33.99
Cranston Manor	$34.95 now	$29.49
Dragon's Eye	$24.95 now	$21.19
Computer Acquire	$20.00 now	$14.99
Twala's Last Redoubt	$29.55 now	$25.39
Alien Rain	$24.95 now	$21.19
Alien Typhoon	$24.95 now	$21.19
Raster Blaster	$29.95 now	$25.39
3-D Skiing	$24.95 now	$21.19
Creature Venture	$24.95 now	$21.19
Galaxy Space War I	$39.95 now	$22.99
Meteoroids in Space	$19.95 now	$16.99
Dragon Fire	$49.95 now	$42.49
Pool 1.5	$34.95 now	$29.49

NEC Green Screen Monitor List $285.00 .... now **$199.00**

Zenith Green Screen Monitor List $159.00 .... now **$109.00**

Ulysses List $34.95 .... now **$29.69**

Speed/asm List $39.95 .... now **$33.89**

Lisa 2.5 List $79.95 .... now **$67.89**

Dinosaurs List $15.00 .... now **$12.69**

BMP Joystick

Here's a joytsick you won't want to be without. It's the only joystick we know about that allows you to change the axis with a flip of a switch. Only the highest quality parts are used.

A real bargain at

$49.95

Apple®

ORDER BY PHONE
800-344-4111 (outside Calif.)
800-692-4143 (inside Calif.)
209-992-5411 (foreign or local)

HUNTINGTON COMPUTING
Post Office Box 787
Corcoran, California 93212

VISA

master charge

We take MasterCard, VISA or American Express (Include card # and expiration date). California residents add 6% tax. Include $2.00 for postage. Foreign and hardware extra. Send for free catalog. Prices subject to change.